Engineering Context-Aware Systems and Applications: A survey

Unai Alegre[a,*], Juan Carlos Augusto[a], Tony Clark[b]

[a]*Research Group On the Development of Intelligent Environments (GOODIES), Middlesex University, London, United Kingdom*
[b]*Department of computing, Sheffield Hallam University, Sheffield, United Kingdom*

Abstract

Context-awareness is an essential component of systems developed in areas like Intelligent Environments, Pervasive & Ubiquitous Computing and Ambient Intelligence. In these emerging fields, there is a need for computerized systems to have a higher understanding of the situations in which to provide services or functionalities, to adapt accordingly. The literature shows that researchers modify existing engineering methods in order to better fit the needs of context-aware computing. These efforts are typically disconnected from each other and generally focus on solving specific development issues. We encourage the creation of a more holistic and unified engineering process that is tailored for the demands of these systems. For this purpose, we study the state-of-the-art in the development of context-aware systems, focusing on: A) Methodologies for developing context-aware systems, analysing the reasons behind their lack of adoption and features that the community wish they can use; B) Context-aware system engineering challenges and techniques applied during the most common development stages; C) Context-aware systems conceptualization.

Keywords: Context-Aware Systems Engineering, Context-Aware Computing, Context-awareness, Context-sensitive, Sentient Computing, Pervasive & Ubiquitous Computing, Intelligent Environments, Ambient Intelligence, Software Engineering, Systems Engineering.

1. Introduction

The miniaturization process of electronics has made a wide range of small devices available with sensing and computing capabilities, taking the computational paradigm out of the desktop. This has opened up new possibilities for interacting with technologies, bringing them closer to people's daily life experiences. The vision of Mark Weiser [1], predicted a trend in which computers disappear by becoming embedded in our daily lives. His view has influenced research in new emerging areas such as Pervasive & Ubiquitous Computing, Intelligent Environments or Ambient Intelligence. The systems developed in these fields need to recognize the context in which they are executing. In this way, they can understand better the situations in which the user expects services delivered and in which way. The aim is to minimize the user effort, enhancing the usability and enabling a better human-computer interaction. Nevertheless, the expectations on context-aware systems (C-AS) can differ from their real abilities [2]. There is a significant contrast between the development needs of systems with contextual awareness, compared to the traditional ones. The literature shows a constant modification of conventional development techniques and methods, to make them suit the C-AS development demands. The challenges of C-AS development are diverse and complex, provoking these techniques to be commonly disconnected from each other, and focused on solving specific issues. The evidence suggests that there is a need of unifying research for developing C-AS. The aim of this survey is to provide better understanding on the basis for an engineering process that is tailored to the demands of C-AS. Our intention is not to provide a whole new engineering process, however, we hope the findings of this article may encourage and inform such future steps within the community. Our research is based on a literature review and the results of a questionnaire carried out to a total of 750 researchers[1]. The remainder of this paper is as follows: *Section 2* presents the state-of-the-art in context-aware systems and its conceptualization. In *Section 3*, challenges and techniques used for C-AS development in each of the most common stages of a system development process. Then, several methodologies are reviewed in *Section 4*, comparing their methods and tools. We analyse the reasons behind the lack of acceptance in methodologies, as well as the features that could have better acceptance. We conclude in *Section 5*, suggesting new directions for Context-Aware Systems Engineering (C-ASE).

*Corresponding author. Tel.: +447871513023
Email addresses: u.alegre@mdx.ac.uk (Unai Alegre), j.augusto@mdx.ac.uk (Juan Carlos Augusto), t.clark@shu.ac.uk (Tony Clark)

[1]The participants were selected from seven conference proceedings between 2011 and 2014: CONTEXT 2011/2013, AmI 2011/2012/2013, IE 2011/2012/2013/2014, UbiComp 2011/2012/2013, Pervasive 2011/2012, IoT 2012, ICCASA 2012/2013. From these, 280 papers were selected as potentially containing researchers with some experience in context-aware computing. A list of 750 names of context-aware systems developers was gathered from the papers and used for contacting the contestants.

2. Context-aware systems

This section focuses on the state-of-the-art in the conceptualization of context and context-awareness for the purpose of developing C-AS. First, we study the basic concepts, identifying a lack of agreement in their definition. Second, we analyse more in depth the causes behind that lack of consensus. Finally, we consider the conceptual limitations of C-AS to characterize their features and interaction types. Through all the section, for illustration purposes, we include an example of a context-aware smart-phone that is able to detect when is intruding into social situations in order to avoid unnecessary interruptions [2].

2.1. Context and context-awareness

In order to implement systems that are able to use the implicit situational information, there is a need to understand the concept of context [3]. This subsection briefly analyses what the community understands by "context".

2.1.1. Context

Although "context" is a term that most people tacitly understand, they find it difficult to elucidate [3]. Many multidisciplinary areas use context to enhance their possibilities. Each area understands the notion as a reflection of its own concerns, making it difficult to define [4]. In the literature, several definitions can be found [3] [4] [5] [6] [7] [8] [9]. A detailed comparison between the differences and similarities of these is out the scope of this survey. Nevertheless, it has to be acknowledged that there is no consensus on the definition of context. Also, we highlight that Dey's [10] is the most acknowledged one, considering it as *"any information that can be used to characterize the situation of an entity"*, where *"an entity can be a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves"*.

2.1.2. Categories of context

Not only is difficult to reach an agreement in what context is, but also on how can it be categorized. Many authors have introduced different context categories and taxonomies [9] [11] [12] [13]. Perera et al. [9] recently presented a broad comparison between them, including their relationship, advantages and disadvantages. They acknowledge two different types of categorization schemes: operational and conceptual. While the operational one helps to understand the issues and challenges of data acquisition techniques, the conceptual allows an understanding of the relationships between different contexts. After comparing all the different categorization schemes, they acknowledge that there is no single one that can accommodate all the demands for context-awareness in the internet of things paradigm.

2.1.3. Context-awareness

Dey [10] also defined a system as context-aware if "*it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task*". Independently from this definition, the adjective "context-aware" is generally used in the literature to describe any type of system that is able to use context. Also, systems that have characteristics which could be considered as "context-aware" use other terms (*e.g.*,"smart", "intelligent") or do not mention any. For instance, let us think about the feature of smartphones that changes the orientation of the screen (landscape/portrait mode) depending on the phone position. Although it could be considered as "context-aware", is typically known by other terms. Also, recently created applications such as Google Now [14] use different terms for this kind of features. On one hand, the great variety of systems and features to be considered "context-aware" make very difficult to make a definition that is suitable for all of them. On the other hand, since "context-awareness" relies on the definition of context, and since there is no consensus on its definition, it is very difficult to characterize what contextual-awareness is.

2.2. The challenges of context

The lack of agreement in the conceptualization of context and context-awareness is just the superficial evidence of a much deeper issue. This subsection studies the ins and outs of context, explaining why there is no consensus and identifying some limitations of these systems.

2.2.1. Context, human activity and human behaviour

Although primitive C-AS can be relatively easy developed (*e.g.*, weather display depending on user location), there is much more potential in applications that involve a deeper context consciousness. The ambition is to create computational systems that are able to understand not only simple contexts, but others such as their environment, the person that is using the device, or the social context in which they are executing services. The deeper conciousness to be implemented, the greater understanding it requires on the complex relation between context, human activity and human behaviour. Following, we briefly analyse some theories on this relation:

> *Situated Action:* Suchman [15] acknowledged that computer artefacts are build relying on an underlying conception, based on the planning model of human action. She introduces the Situated Action, an alternative in which it is analysed how people find meaning in actions or how should they construct it. Instead of producing formal models of knowledge and action, she proposes exploring their relationship to the particular circumstances in which they occur. The unit of analysis of Situated Action is not the individual, not the environment, but a relation between the two [16].
>
> *Activity Theory:* Claims that context is defined by the activity itself. *"Activity comprises a subject (the person or group doing the activity), an object (the need or desire that motivates the activity), and operations (the way an activity is carried out). Artefacts and environment are seen as entities that mediate activity"* [17]. The unit of analysis in this theory is an activity [16].

Distributed Cognition: Takes into account the representation of knowledge both inside the heads of individuals and in the world [18]. The system is not considered relative to an individual but to a distributed collection of interacting people and artefacts [16]. Hence, the unit of analysis in this theory is the whole system, centring in its functionality and understanding the coordination among individuals and artefacts [16].

The Locales Framework: Tries to understand the nature of social activity and work, and how *locale* can support these activities [17] [19]. *Locales* are considered as a social worlds that apportion and use particular locations and means for accomplish work. These, are abstract, and do not necessarily need to have a fixed meaning or be associated to a physical space.

Ethnomethodology: Focuses on the way people make sense to their everyday world, capturing a range of phenomena associated with the use of mundane knowledge and reasoning procedures.

Initially, the reader could think that embedding a deeper contextual awareness into systems is just a matter of understanding the context through one of these theories and then creating a programmable model with the result. But the fact is that these theories work differently. They all share an understanding of social facts as having no objective reality beyond the ability of individuals and groups to recognise and orient towards them. Conversely, the developers of C-AS will naturally seek to reduce complex observable phenomena to essences or simplified models that capture underlying patterns, abstracted from the detail of particular occasions. These models, try to seek an objective reality in social facts, entering in conflict with the foundations of most theories in social analysis, which are incompatible with the idea of a stable external world that is unproblematically recognized by all. This issue was recognised by Dourish [5]. He acknowledged that the problem comes from the overlapping of two philosophical traditions behind the understanding of context: Positivism and phenomenology. Context-aware computing stems from computer science, that derives from the rational, empirical and scientific tradition of positivism. On the other hand, phenomenology, is the background behind many of the theories to explain context in complex human behaviour and activity. The incompatibilities between these standpoints help to explain what are the limitations that exist when developing C-AS, as it is further described in the next subsection.

2.2.2. The limits of Context-Aware Systems

Once that the reader is aware that there are tensions between two incompatible philosophical backgrounds, s/he can start to understand why is so difficult to find an agreement on the definition of context. In what regards to its conceptualization, phenomenology recognizes context as an interaction problem, considering that: "*(I) Context is particular to each occasion of activity or action; (II) The scope of contextual features is defined dynamically; (III) Context may or may be not relevant to some particular activity; (IV) Context arises from activity, being actively produced, maintained an enacted.*" [5]. In this approach, the context can only be understood as the situation arises. Then, there is no need to unearth the underlying models that will describe the objective reality behind context, alleviating developers from the task of having to foresee it. For making this possible, there is a need to make machines exhibit human-like cognitive skills. The idea is to extract the mathematical model of a brain, imitate it in a computer, and train it to satisfy the user needs [20]. Nevertheless, it has to be mentioned that there has been a long debate about the feasibility of computational systems mimicking human-like intellect since the early origins of artificial intelligence [21] [22] [23] [24] [25]. Although it is not the purpose of this survey to analyse in depth these theories and argue about them, we would like to stress that they acknowledge some issues that artificial intelligence and context-aware computing have not been able to solve yet. For example, the limitation of computers to acquire expertise in the same degree and areas as humans, due to their different form of embodiment [22]. What is important for the reader is to understand that the contextual awareness of machines is from a radically different nature than the one of humans. Also, that computational systems are good at gathering and aggregating data, but humans are still better at recognizing contexts and determining what action is appropriate in a certain situation [2].

On the other hand, positivism looks at context as a representational problem, considering it as a "*form of information, delineable, stable and separable from activity*" [5]. The definitions made in the context-aware field, naturally adopt this point of view. For instance, Dey's definition [10] allows designers to use the concept for determining why a situation occurs and use this to encode some action in the application [26], making the concept operational in terms of the actors and information sources [17]. Nevertheless, since the definition inherently has a positivist view, the potential of C-AS remains limited to the context that developers are able to encode and foresee. Let us retake the example of the context-aware phone that is able to silence itself at certain situations. For instance, let us imagine that developers want to detect that the user is at the cinema. For this task, they can use the built-in sensors of the phone to detect luminosity, location, motionlessness and ambient sound. Then, they can program a rule to silence the phone whenever its sensors indicate certain values. In this way, the phone can detect when the user is at the cinema, but for detecting a completely different situation, the sensor values that will silence the phone (or even the sensors used) could be completely different. The problem of having to program computers, is that developers must know beforehand the context that they need to program. But they might not be able to foresee some situations. The user might be at a job interview, s/he might be sleeping when some irrelevant notifications or calls arrive, the user could be in the middle of a wedding, funeral, trial, having a very important conversation, in the library, etc. The list of unforeseen or undetectable contexts can be endless. Besides, some situations can be specific issues of one user, or may not be useful enough to carry the effort of implementing them (*e.g.*, they may just happen once in the whole system life-cycle).

Summarizing, if developers can not determine all that can be affected by an action, it will be very difficult to write a closed and comprehensive set of actions to take in those cases. There are three tasks that a developer may find difficult, or even impossible, when developing a C-AS under this perspective [17]: (A) Enumerate the set of contextual states that may exist; (B) Know what information could accurately determine a contextual state within that set; (C) State what appropriate action should be taken in a particular state.

As it can be observed, the development of C-AS is inherently in conflict with two opposite philosophical paradigms. Due to the limitations in both approaches, the near future of C-AS development should not be seen directly towards creating exclusively systems that exhibit human-like contextual awareness, nor to exclusively programming them on the basis of foreseen context. The creation of C-AS in the near future, comes through a combination of the current advances in both approaches, providing a higher cooperation between humans and computers and making the most of each others qualities.

2.3. Towards the characterization of context-aware systems

In this subsection we focus on the aspects characterizing context-dependent applications in order to better engineer them. Taking into account the influence of C-AS limitations, we explore the ways in which C-AS can interact with users, categorizing them. Finally, with the interaction categories in mind, we characterize the features of these systems.

2.3.1. Interacting with a Context-Aware System

Inspired in the human-like contextual awareness, C-AS were originally intended to monitor the context and then act accordingly without any human mediation. The aim of having an autonomous system is to reduce the user intervention, easing its use and decreasing user distraction [27]. As discussed in the previous section, humans are fitted with better contextual understanding capabilities. So, when a system takes away the user control due to a misinterpretation of the context, in situations where the user has a better understanding of what is happening, the user can reject the system and stop using it. In order to alleviate this problem, other viewpoints propose to change the autonomy of C-AS, enabling the users to have more control over the system actions. Let us retake the auto-silence smartphone example. Instead of letting the phone itself decide when to silence, a machine could answer when someone is calling [2]: "Lee has been motionless in a dim place with a high ambient of sound for the last 45 minutes. Continue with the call or leave a message?". In this way, the higher understanding of context that humans naturally have, can be used to complement the decision making about the system actions, depending on the situation. As readers can observe, there are different ways to interact with C-AS. Barkhuus et al. [28] classified them into: A) Personalization, in which the users are able to set their preferences, likes and expectations to the system manually [9]; B) Passive context-awareness, where the system is constantly monitoring the environment and offers choices to the users in order to take actions; C) Active context-awareness, where the system is continuously monitoring the environment and acts autonomously.

We have classified the interaction with C-AS in two different modalities: Execution and configuration. The first one refers to the actions/behaviours of the system when a specific situation arises (*e.g.*, auto-silence smartphone receives a call at 4 a.m. and decides to silence it). The second one is related to the adjustment of actions/behaviours that a system will be exhibiting in the future (*e.g.*, according to the preferences of a certain user, the phone is configured so that future calls coming at 4 a.m. are not automatically silenced). Both execution and configuration modalities are independent between themselves, but they can both have a degree in between: I) Active, where the system changes its content autonomously; and II) Passive, where the user has explicit involvement in the actions taken by the system. Following, we analyse them more in depth:

Active Execution: In this interaction type, the systems act autonomously depending on the context in which they are embedded. For example, the screen of a smart-phone can switch from landscape to portrait automatically, depending on the values of its accelerometer. The heater in a smart-house can be autonomously switched on and off when the values of a thermometer sensor reach a certain point. In this approach the vision of self-adaptive systems is paramount, which are able to adjust their behaviour in response to their perception of the environment and the system itself [29] [30]. Mizouni et al. [31] presented a framework for context-aware self-adaptive mobile applications using the advantages of the software product line feature modelling to manage variability. Projects such as MUSIC [32] [33] [34], also support the development of self-adaptive systems in ubiquitous environments.

Passive Execution: The users are involved in the action taking process of the system, where they specify how the application should change in a specific situation [35]. The system can present available services for that specific situation or ask for permission to take an action. The user can also receive additional information of the context that can support their decision taking, or cues about why the system is behaving in a certain way. Dey and Newberger [36] encourage the use of intelligibility features to let the user control the system. Those techniques can help expose the inner workings and inputs of context-aware applications that tend to be opaque to users due to their implicit sensing and actions [37]. It allows understanding how a context-aware application is working or behaving by showing it and can be used to allow a better user control. Lim and Dey [38], present the intelligibility tool-kit to give support for context-aware applications. They facilitate developers to obtain eight types of explanations from the most popular decision models of context-aware applications.

Active Configuration: In this interaction type, the system is able to learn from the user preferences in order to autonomously evolve his rules for future behaviour, after

the system is implemented. Mori and Inverardi [39] [40] present a software life-cycle process for context-aware adaptive systems, where they characterize context by foreseen and unforeseen variations. In the first case the system evolves in order to keep satisfied a fixed set of requirements while in the second one the system evolves in order to respond to requirements variations that are unknown at design-time. In a later work [41], they focus on a decision support mechanism for simultaneous adaptation to system execution context and user preferences. Aztiria et al. [42] introduce a system which is able to discover patterns in the user actions to learn their frequent behaviour when interacting with Intelligent Environments. The system can generate automatically context-aware reasoning rules [43].

Passive Configuration: The user is involved in the manual personalization of his/her preferences, likes, and expectations of the system, after its implementation. By reducing the complexity of programming, it enables the system behaviour configuration to inexperienced users. These, acting as non-professional developers, can create, modify, or extend existing context-aware artefacts. Lieberman et al.[44], originally introduced this approach, classifying the type of activities[2] involved in it as: A) Parametrization or customizations, considered as activities that allow users to choose among alternative behaviours already available in the application; B) Program creation and modification, in the form of activities that imply some alteration, aiming at creating from scratch or modifying existing software artefacts. In what regards to C-AS, the Trigger-action programming[3] [45] [46] is recently gaining more popularity. In this approach, the end-users can handle simplified if-then programming rules that match a trigger with an action. Dey et al. [47] proposed *iCap*, a system that is the intermediate layer between low level tool-kits and users. They also present a specific solution [48] for user control, based on their Context Tool-kit [49].

C-AS do not necessarily have to be completely active or passive, they can have hybrid approaches with different degrees in between. For example, the autonomy level can be adjustable, enabling human users to collaborate with computational systems managing the system behaviour as a team. Ball et al. [50] consider enabling human-agent teamwork in Intelligent Environments by employing concepts of adjustable-autonomy and mixed-initiative interaction. Such approaches reduce the chance of guesswork needing to be done. If the user or agent can not manage the system in the usual way, they can seek help from each other, inheriting the benefits from autonomy level that is implemented. On the other hand, it also inherits the drawbacks from its opposite autonomy level. The advantages and disadvantages of the different interaction categories are presented in Table 1.

[2]Further information about the different ways of customization can be found in Lieberman et al.[44].

[3]More information about Trigger-action programming can be found in Section 3.4.2.3.

2.3.2. Features of a Context-Aware System

Schilit et al. [13] first identified different classes of context-aware applications. Pascoe et al. [55], later aimed at identifying the core features of context-awareness. Dey and Abowd [3] presented a categorization for features of context-aware applications, based on the classification of Schilit and Pascoe, namely:

1. *Presentation of information and services to the user.*
2. *Automatic execution of a service.*
3. *Tagging of context to information for a latter retrieval.*

The first feature decides which information and services are presented to the user, based on context. Nearby located objects might be emphasized, or for instance, a printer command might print to the nearest printer. The second feature is the automatic execution of a service. For example, let us consider a smart home environment. "*When a user starts driving home from their office, a context-aware application employed in the house should switch on the air condition system the coffee machine to be ready to use by the time the user steps into their house*" [9]. Finally, they present "contextual augmentation", which extends the capabilities of sensing, reacting and interacting with the environment by using additional information. This is achieved by associating digital data with a particular context. For example, a tour guide can augment reality by presenting information about the attractions that they are surrounded by or are approaching [55]. In the previous subsections, we discuss the need to: A) Take into account the current limitations of C-AS; B) Include the different interaction levels; C) The main challenges that are still open in the context-aware computing field. In order to accommodate these demands, we propose to extend Dey and Abowd's features of C-AS into:

1. *Presentation of information to the stakeholders*
2. *Active or passive execution of a service*
3. *Active or passive configuration of a service*
4. *Tagging context to information*

The first one is very similar to Dey and Abowd's. It keeps the essence of Pascoe's "presenting context", but Schilit's "proximate selection" and "contextual commands" are merged with our second feature. We have introduced the notion of collaboration among stakeholders rather than just the users. We can have different "situations of interest" according to the category of stakeholders (*e.g.*, for a primary user it could be that the system is aware on the weather forecast, but for the engineers it may be that the system is aware of the level of battery in the mobile phone). So, we can differentiate Primary users' contexts, Secondary users' or system engineers', among others. Some contexts of interest from all these stakeholders will typically intersect. They do not necessarily have to be disjoint, equally they do not have to be exactly the same, there are no a-priori relations and it all depends on the applications and personal choices. Even when some "situations of interest" may be the same for different stakeholders, they may be interested in them for different reasons and may expect different outcomes from the system when those situations materialize. The contextual information might arrive to some secondary or tertiary users

Name	Pros	Cons	Name	Pros	Cons
Active Execution (Self-adaptivity)	• Little or no effort required by users [51] • No special user knowledge is needed [51]	• Difficult to ensure that the system will take an appropriate action (Difficult to validate and verify the system) • Loss of control over what the system is executing and why [51] • There are still some open issues [29] [52] • Developers have all the burden • Users can be uncomfortable not understanding what happens with the information that the machine gathers from them	*Passive Execution* (Intelligibility & Control)	• Augments the trust of users [53] since they understand better how the system works • Easier to evaluate the system behaviour • The system will take the actions that the user wants	• Requires developers to understand how to generate explanations [38] • The users might not have enough expertise to take decisions on their own • Applications need to convey more information to explain actions to users [53] • May compromise the privacy of users if they are used on social interactions • Users can use their higher context understanding for a better control the system [53] [2]
Active Configuration (Learning & Adapting)	• Little or no effort required by users • No special user knowledge is needed • Can unearth needs, preferences or habits difficult to see in other ways [42]	• Difficult to determine when rules should be created or deleted • The rules are based on sensors values (inaccuracy and uncertainty) • Loss of control over what the system is executing and why [51]	*Passive Configuration* (End-user programming)	• Offers greater motivation, control, ownership, creativity and quality to end-users [54] • Users are in control; users know their tasks best [51] • Releases developers burden	• Users might be forced to contribute and cooperate in context for which they could lack experience [54] • Meta-design is more complex and abstract than design • Complexity is increased (users need to learn adaptation components); Systems may become incompatible [51]

Table 1: Comparative analysis on the interactivity levels that context-aware systems can have.

that make the choices over the actions of the system, based on the users needs. Such as what happens in projects like POSEIDON[4] [56], where there is a need of secondary users (*e.g.*, parents) to take care of the primary ones (*i.e.*, person with Down's Syndrome). Our second feature includes all the different involvement degrees of the user in the system actions, as the situation arises when it is executing. A service can be automatically triggered, being the system autonomous in its decision. But it also can ask the approval of the user, or display a certain list of possible choices, as in Schilit's "proximate selection", to enable further collaboration between the system and the users. The third feature is related with being more useful to the stakeholders, relating its services to their preferences and needs, which can evolve in time. C-AS can adapt to these through the active or passive configuration of the system, as explained in the previous subsection. Finally, the last feature is the same as Dey and Abowd's.

3. Context-aware systems development techniques

We have investigated the state-of-the-art in C-AS, understanding better its conceptualization. In this section, we first analyse the challenges of building such systems. Second, we study the C-AS development techniques in the literature throughout the most common stages of a development process: Requirements Elicitation, Analysis & Design, Implementation and Deployment and Maintenance.

3.1. Development Challenges

The following subsection highlights the challenges and demands of C-AS development. We focus more in the context information handling, as it is the most important and complex need. We also describe issues related to the diversity of systems and other important technological demands.

3.1.1. Diverse and specific systems

Almost any type of computerized system could potentially benefit from having contextual understanding. Although different kind of systems can be found next to the adjective "context-aware", the process of embedding contextual awareness can be dramatically different depending on the system type. For example, it is not the same to implement it in: a smart-phone application, a robot, a ubiquitous system or a web application [57], a context-aware animal species recognition [58], or an adaptive e-book [59], etc. Typically, context-awareness is a feature added on top of an existing system or functionality. Then, the implementation of such a feature depends intrinsically on the system where it is going to be implemented. Let us take the example of the context-aware smart-phone that is able to detect when is intruding into social situations in order to avoid unnecessary interruptions. In this case, the features will be developed on top of the phone, its operating system, and its application to make calls. This fact turns the system into a very ad-hoc solution. The main problem of being so specific, is that the amount of work employed to develop it will be difficult to reuse, even for developing the same for a different operating system.

3.1.2. Context information handling

In order to enable context-awareness, there is always a need of capturing context information and making it available to applications and systems [60]. C-AS require separating how context is acquired from how it is used, so that applications are able to use contextual information without knowing the details of a sensor and how can it be implemented [49] [61]. The techniques for context information management have been widely

[4]POSEIDON stands for PersOnalized Smart Environments to increase Inclusion of people with DOwn's syNdrome

researched and are well understood [9]. Despite the advances, the challenge for an engineering process is to facilitate the development and reuse of structures that enable context information management and support the adaptation to the specific needs of the applications/systems. Following, the life-cycle [9] of context information is used as a reference to better clarify the issues that this information management may entail:

Acquisition: First, the context information needs to be gathered. Generally, this happens from multiple and distributed resources, which makes the quality and authenticity of information difficult to achieve. On the other hand, sensors in general, are likely to provide inaccurate, overlapping, contradictory or missing data (*e.g.*, providing the same information at different timings or with a jitter) [62]. It also has to be mentioned, that the addition and removal of context resources can give rise to scalability issues. Finally, it has to be taken into account that it will be difficult to obtain contextual information if many users share the same physical sensors and service resources [63].

Modelling: After the information is sensed, it needs to be translated into usable values. In this process, real world concepts are translated into modelling constructs. A raw value of a user position may be in the form of: "42.85, -2.683333". This information must be translated into more understandable information such as the name of a city, street, etc. These models require [64] [61]: 1) To represent any kind of context information, reflecting the entities of the real world and their relations; 2) Uniquely identify the contextual information, context and entities; 3) To be simple, reusable, expandable and able to use the information at runtime; 4) Validate pieces of data and encode its uncertainty.

Reasoning: Based on the modelled data, different kinds of conclusions can be inferred, where this data can be seen as evidence to support the conclusion [65]. In this way, new knowledge and understanding is obtained, based on the available context [66]. . This process has typically three different phases [9]: (1) Context pre-processing, where data is cleaned to get rid of invalid, inaccurate and non desirable values; (2) Sensor data fusion, in which sensor data is combined to produce more accurate and dependable information; (3) Context inference, from low-level information to high-level one. In what regards to reasoning, representation expressiveness is in mutual conflict with soundness, completeness and efficiency [67].

Dissemination: Finally, both low-level and high-level context need to be distributed to the consumer. The context information must have high availability, ideally to be provided it in real-time. Another desirable feature is to discover new services that could provide new context information [65].

It also has to be mentioned that context information will inherently contain important data related to the users, what raises some privacy issues. Privacy concerns may differ from user to user, and may also be dynamically changing over time. The balance between privacy and the system potential is delicate, where the developer may fall into ethical issues. A detailed examination of these issues is not the focus of this survey, but the reader can have more information about ethical concerns that can influence the area at engineering level in [68]. Besides, it is difficult to obtain, ensure and evaluate the good Quality of Context (QoC) information, which depends on its [69]: Precision, probability of correctness, trustworthiness, resolution and contemporaneity.

3.1.3. Technological demands

The technological challenges are not only focused on the information management. We have classified the remaining relevant needs of C-AS systems into:

- *Flexibility versus change:* Once the context information is provided, the rest of the system configuration can happen in many forms, that depend on the specific implementation of the system itself. Although the particular implementations can vary, a need that seems invariant is supporting a high amount of changes.
- *Cost:* C-AS are expensive to develop, deploy, execute and maintain. The amount of information that they need to manage makes them resource hungry [70] and dependable on a very expensive structure.
- *Reliability:* C-AS must be reliable, especially if they are going to be used in tasks where an error can put a human life at risk. Even if they are not able to offer a continuous delivery of some services properly, they at least should be capable to perform its required basic functions tolerating errors, faults and failures. Fault tolerance in pervasive systems can be increased by [71]: (1) Efficiently detecting faults; (2) Isolating faults, to prevent its propagation to other parts of the system; (3) Providing a transparent fault tolerance; and (4) Good fault reporting mechanisms. Besides, it is difficult to evaluate the correctness of C-AS due to their increasing size and device diversity.
- *Infrastructure:* The expensiveness and complexity of C-AS development makes highly desirable for developers to have tools that support and ease their effort during the development of their systems. An infrastructure [72], is software that supports construction or operation of other software, comprising systems that range from tool-kits to network services or other sorts of platforms. So, it enables applications that could not otherwise be built or would be prohibitively difficult, slow or expensive. This kind of infrastructures are typical in C-AS development. Nevertheless, a certain infrastructure affords only certain styles of application and interface. This creates a tension between easing the development and the flexibility of the infrastructure, and it is a challenge itself.

Although the field has matured in the latest years, developers have still many issues to overcome when developing C-AS. The next step is to fathom the techniques proposed to develop this kind of systems.

3.2. Requirements elicitation

The requirement elicitation process helps developers to reach a better understanding of the user needs and demands by finding a systematic approach for eliciting, analysing, documenting, validating and managing software requirements from individual stakeholders [73] [74] [75]. If the right requirements are not well defined prior to the development of the system, it will be more likely to fail meeting the user and other stakeholder's expectations. Some conventional techniques for eliciting requirements can be classified into [76]: Interviews, questionnaires, task analysis, domain analysis, introspection, repertory grids, card sorting, laddering, group work, brainstorming, joint application development, requirements workshops, ethnography based, observation, protocol analysis, apprenticing, prototyping, goal based, scenario-based and viewpoints. The aim of this subsection is not to deeply review them, but to focus in requirement elicitation techniques that have been tailored for meeting the needs of C-AS development. We have classified them into: A) Context categorizations; B) User-centred and Social Science based; C) Requirements modelling D) Adaptive and goal oriented. We compare their advantages and disadvantages in Table 2. Finally, we analyse previous works for identifying common features of requirements elicitation techniques for C-AS, to which we add our conclusions.

3.2.1. Categories of context in Requirements Elicitation

Some requirements engineering techniques are based on context-classifications. Krogstie [77] studies the challenges that specify the requirements to applications running on mobile technology. He presents six context categories (i.e., spatio-temporal, environment, personal, task-oriented, social and information contexts) to guide the design of customised stakeholder interviews. Hong et al. [78] present a methodology for requirements elicitation in context-aware ubiquitous application design. In their approach, they propose to use the notion of extended context, categorizing it into: A) Computing context, referring to the hardware configuration used; B) User context, that represents all the human factors such as user's profiles or calendars; and C) Physical context, that considers the non–computing-related information provided by a real world environment. Kolos-Mazuryk et al. [79], based on [35] [13], present a classification of contextual and non-contextual distinguishing properties of ubiquitous applications. On the contextual properties, they acknowledge C-AS as having: I) Dynamic environment; II) Variable bandwidth; III) Changing display characteristics; IV) Changing user environment; V) The target platform is not known in advance. Finally, they present an initial phase of a requirements engineering process.

3.2.2. User-Centred and Social Science based approaches

Sutcliffe et al. [80] introduce a Personal and Contextual Requirements Engineering framework (PC-RE), a method for their capture that incorporates a trade-off analysis to decide how personal requirements should be implemented. They provide three different layers: Stakeholders, user characteristics and personal goals. These, can change over space and time. In what regards to requirements elicitation for C-AS development, they consider the locales conceptual framework [19] for including spatial implications in requirements. They acknowledge that monitoring requirements for this kind of applications needs to be specified for: A) The devices and sensors that capture environmental input; B) Functions and processes that interpret low-level data into streams of meaningful data; C) Interpreters that make sense of the data using a model of the domain; D) Models of the domain. They acknowledge that the results of interpreters (*e.g.*, body temperature above a certain threshold is dangerous for a patient) are frequently ambiguous so further requirements need to be considered: I) Intelligibility feedback functions; II) Default interpretations when data input is not available or is inadequate; III) Mediation dialogues should be planned to enable users override systems' decision.

Kjaer [81] proposes a requirement gathering process for the design of context-aware middleware. They video-recorded and documented the activities of people who worked at a farm while they were doing their daily work. Using an ethnographic study to classify the context, they determined requirements for the middleware they were trying to develop. Evans et al. [82] present R4IE, a framework for a requirements engineering process for Intelligent Environments, in which context-awareness is a primary feature. Their work is similar to Sutcliffe's but they include stakeholder profiling with individual user customization. They also introduce a core ethical model, enhancing the addressing of the issues of social context and ethnicity, considering privacy.

3.2.3. Requirements Modelling

Desmet et al. [83] present Context-Oriented Domain Analysis (CODA), a systematic approach for gathering requirements of C-AS. It enforces modellers to think by means of "context-unaware" behaviour, which can be further refined according to "context-aware dependant adaptations" at certain "validation points". They also identify existing relationships between context-dependent adaptations. CODA can be represented: in a tree structure (graphically), using XML for writing its diagrams (textually) and mapping its semantics to elements in the decision tables (structurally). Choi et al. [84] propose a method for requirements gathering in C-AS based on variations of UML Use Case Diagrams. They classify context-aware services in five types, in order to give stakeholders a better understanding when analysing C-AS. They also introduce a process for requirements analysis, context-aware use case diagram, context-switch diagram and dynamic service model for context-aware systems. In a further work [85], they introduce decision tables or trees. They encourage analysts and stakeholders to pay attention to context related issues such as system platform, target users, intelligence, possible context-aware services and agreement with other stakeholders as well as understanding context with decision tables and trees. Ruiz et al. [86] describe a model-driven engineering approach targeting non-functional requirements, where they: A) Derive a model-driven system design that meets specific requirements; B) Generate code that implements such design. Sitou and Spanfelner [87] present RE-CAWAR a requirements engineering process for context-aware

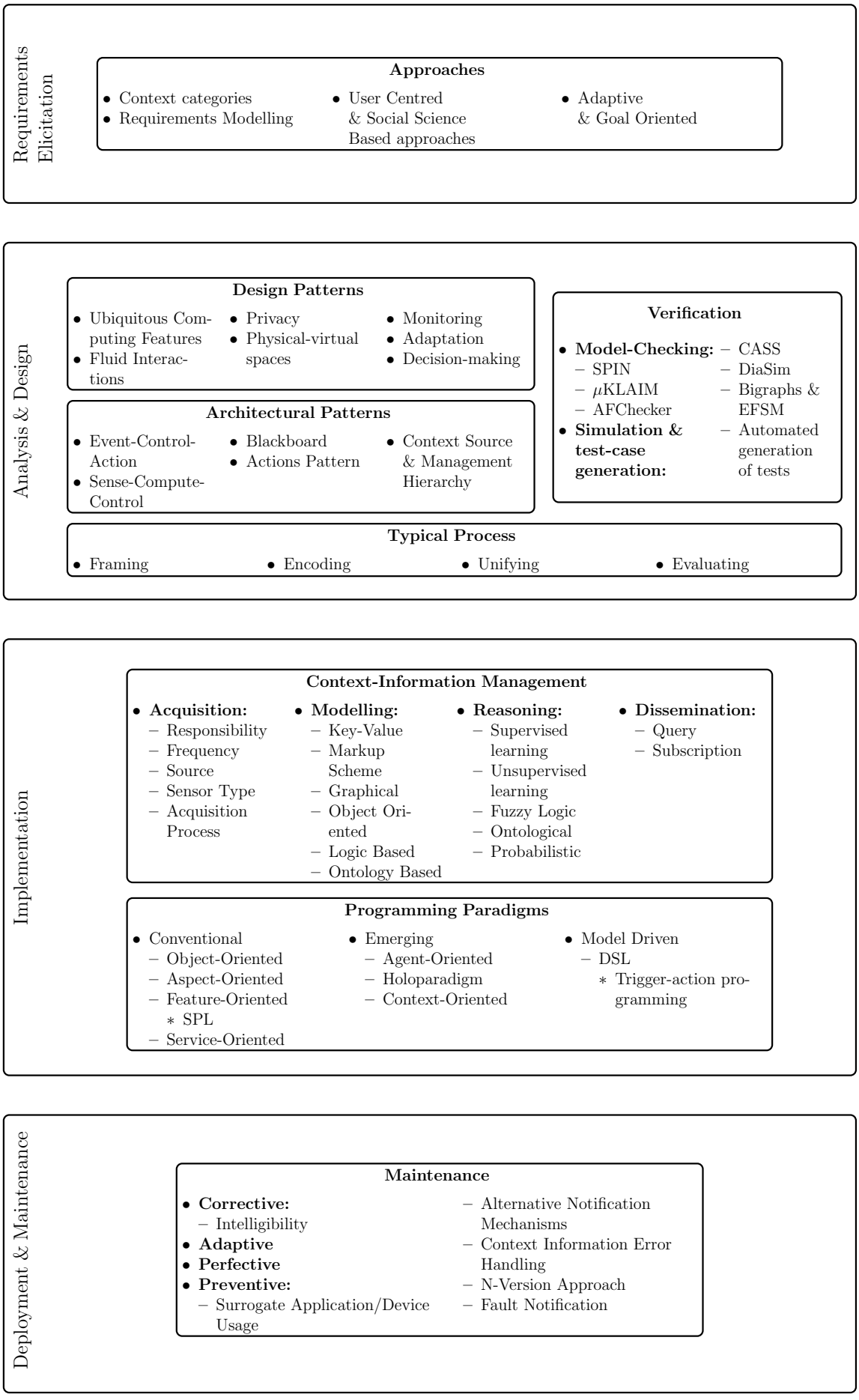

Figure 1: Summary of the techniques/approaches used for Context-Aware Systems Engineering.

and adaptive systems. It provides an integrated model for the usage context based on different models: I) User Model, where the participants aspects are represented, characterizing the users and user groups; II) Task Model, to represent the activities aspects, identifying which task and interactions are needed to perform it; III) Domain Model, that consists of any user visible, operable objects in the applications' domain, representing the operational environment aspects; IV) Platform Model, that represents the physical infrastructure and the relationship between the involved devices; V) Dialogue model, where the interaction between the user and the system is depicted; VI) Presentation model, that shows visual haptic and audio elements needed for the interaction. Through an iterative process, stable needs are identified, as well as such that may change according to the context. The stable needs result in the functionality of the system, while the situational needs are further analysed to specify the adaptation logic.

3.2.4. Adaptive and Goal-Oriented

Finkelstein and Savigni [88] present a framework for requirements engineering in context-aware services, where they propose that requirements themselves can change during the system execution. They difference between: A) Goals, as a fixed objective of the service; B) Requirements, as a more volatile concept that can be influenced by the context. Oyama et al. [89] describe an approach of service requirements analysis using the feedback of contexts, to support the elicitation of user intentions and goals robustly. They identify two approaches in the evolvability of C-AS: I) Short-term evolution, to handle exceptions and to make correct reactions at runtime; II) Long-term evolution, to monitor user behaviour and capture new system requirements based on human intentions. Baresi et al. [90] present FLAGS, a goal model that adds adaptive goals in order to embed adaptation countermeasures, fostering self-adaptation by considering requirements as live, runtime entities. They distinguish between: 1) Crisp goals, whose satisfaction is Boolean; 2) Fuzzy goals, whose satisfaction is represented through fuzzy constraints. Adaptation countermeasures are triggered by violated goals and the goal model is modified accordingly to maintain a coherent view of the system and enforce adaptation directives on the running system. Siadat and Song [91] discuss the state-of-the-art requirements for adaptive systems, under the notion that requirements that are engineered at design time may require further reasoning or refinement at runtime in order to adapt to dynamic context-driven changes.

3.2.5. Conclusions

Other surveys have also studied the requirement elicitation techniques specialized for C-AS development. Following, we analyse their view on the most common characteristics in requirements elicitation for C-AS. Preuveneers and Novais [92] present a survey of the best software engineering practices in Ambient Intelligence. In all the approaches they surveyed for requirements elicitation, they highlight: A) The importance to actively involve the end-user and to develop an elicitation process that is customized to the competences of the end user; B) The need for an explicit representation of the context and goals of the user, and how the context impacts the interaction with the user; C) An explicit formalization of which requirements are relevant for a given context and how the these can evolve when the context change. Alshaikh and Boughton [8] analyse how context works in requirements elicitation. In early stages, context is associated with the task of setting system boundaries, while in later stages, context is used implicitly within the scenario-based requirements (common-sense approach). Finally, they distinguish between context and requirements. *Context* is defined through the user's situation and *Requirements* are described focusing on the user's interaction with the system. Evans et al. [82] classify the prominent themes in requirements elicitation focused on context-awareness for Intelligent Environments: 1) The consideration, adoption and possible enhancement of a context taxonomy; 2) A general assumption that systems need to be adaptable to be context-aware; 3) Elicitation techniques used to capture end-user cognitive tasks require enhancement to account for context-awareness; 4) Identification of target user groups and acknowledgement that contextual requirements for either profile may evolve over time; 5) Requirements themselves might be context-driven; 6) The consideration of cultural context; 7) The adoption of goal-oriented requirements engineering where higher-order goals are apparent in the domain. They also acknowledge that there is lack of standards in what regards to requirements engineering for C-AS. From the literature review, we insight that a requirements elicitation process, tailored to the demands of C-AS, needs to support engineers:

1. Identifying situations in which services that are relevant to the stakeholders could be provided. We acknowledge that

Name	Pros	Cons
Pre-categorized	• Can complement other requirement elicitation techniques	• Categories might be too broad or to narrow • Categorization schemes can not accommodate all the demands for C-AS [9]
Social Science Based	• Complement the SW engineering knowledge with the one of social experts • More complete requirements	• Complexity of including social experts as new stakeholders • Cost
User Centred	• Focuses on the needs of the stakeholders • Improves the usability	• Requires user involvement
Model Driven	• Forces developers to have different views • Help to decide the boundaries between elicitation and design • Enables better traceability	• Different readers can make different interpretations • Hard to capture non-functional requirements
Adaptive & Goal Oriented	• Provides means to capture and analyse variability • Opens the way to requirements evolution	• Difficult to determine when the behaviour of the system meets the requirements

Table 2: Comparative analysis on the advantages and disadvantages of the different requirement elicitation techniques that are specialized for context-aware systems development.

social sciences and social experts could potentially help to understand better these situations.

2. Determining how the system is going to detect those situations. It does not have to be an exhaustive description, but it should facilitate the future design on how the contextual information is going to be acquired, modelled, reasoned and distributed.
3. Choosing how the system will interact[5] with the stakeholders:
   (a) Determining the behaviour that the system will exhibit in those situations.
   (b) Identifying the possible configurations that can help to evolve the system after its implementation.

3.3. Analysis and Design

When the system requirements are well specified, an analysis can ease the development plan through a better understanding of the system implementation. The design brings developers closer to a feasible implementation plan. It has to be acknowledged that there is no set of universally accepted basic design and development principles, or standards, which lead to a uniform approach to the efficient C-AS development. The aim of this subsection is to study the different approaches for analysing and designing C-AS. We first focus on the design process itself, to examine more in depth: A) The different architectures of C-AS and architecture patterns used for design; B) Middleware; C) Design-Patterns; and D) Design evaluation.

3.3.1. Design process

Bauer et al. [93] identified the most common practices used by developers when designing a C-AS, dividing the process into: *A) Framing*: Designers will articulate and explore a concept of context, which imposes a set of limitations on what exist inside and outside the design space their work inhabits [93]. *B) Encoding*: In this stage designers will discuss the behaviour of the system and instantiate a vocabulary or codes to express its behaviour. *C) Unifying*: As the designers explore the design space, certain possible design solutions are brought to the foreground, which impose additional constraints over other concerns the designers address. *D) Evaluating*: The designer will focus on a solution that satisfies the constraints according to their encoded formulation of context, determining if they have arrived at a satisfactory solution.

3.3.2. Architectures

An architecture is an abstraction, that generalizes the systems, without showing detailed implementation such as code or circuits. This subsection studies the architectures used for C-AS development. Due to their diversity, C-AS have adopted disparate ranges of architectures. Table 3 shows the different classifications of architectures according to existing literature surveys. Following, we analyse the works that classify generic architectures and we study the different architectural patterns that can be used for creating C-AS.

[5]More information about interaction categories can be found in Section 2.3.1

Author(s)	Winograd	Chen et al.	Perera et al.
Reference	[94]	[95]	[9]
Year	2001	2004	2014
Architecture Types	• Widgets • Networked Services • Blackboard Model	• Direct Sensor Access • Middleware Based • Context Server	• Component Based • Distributed • Service Based • Node Based • Centralized • Client-Server

Table 3: Different architecture classifications for context-aware systems.

3.3.2.1. Generic architectures. Other works also study the common parts that different architectures have, providing a generic architecture for C-AS. Baldauf et al. [96] present a survey on C-AS, in which they introduce an abstract layer architecture for C-AS that is divided in: A) Sensors; B) Raw data retrieval; C) Storage/Management; D) Pre-processing; and E) Application. Hong et al. [63] introduce a literature review of C-AS, in which they classify architectural layers of these systems into: I) Concept and Research; II) Network infrastructure layer; III) Middleware layer; IV) Application layer; and V) Infrastructure layer.

3.3.2.2. Architectural patterns. An *architectural pattern* is a set of architectural design decisions that are applicable to a recurring design problem, and parametrized to account for different software development contexts in which that problem appears [97]. We have gathered from the literature, the following patterns:

- *Context Sources & Managers Hierarchy [98]*: This architectural pattern aims at providing a structural schema to enable the distribution and composition of context information processing components (Context Sources and Context Managers). A Context Source encapsulates single domain sensors (*e.g.*, blood pressure), while a Context Manager component covers multiple domain context sources (*e.g.*, integration of blood pressure with heart beat measures) [99]. The structural schema consists of hierarchical chains of them, in which the outcome of a context information processing unit may become input for the higher level unit in the hierarchy. It offers a flexible and decoupled distribution of context processing activities (sensing, aggregating, inferring and predicting). It also improves collaboration among context information owners allowing new parties to join the collaborative network in order to provide richer context information.
- *Blackboards [94]*: The architectural style arose in artificial intelligence applications and its intuitive sense is one of many diverse experts sitting around a blackboard, all attempting to cooperate in the solution of a large, complex problem [97]. As an architectural pattern, it adopts a data-centric point of view. Rather than sending requests to distributed components and getting call-backs from them, a process posts messages to a common message board, to which others can subscribe to receive messages matching

a specified pattern that have been posted. All communications go through a centralized server.

- *Event-Control-Action [98]*: It has been devised in order to decouple context concerns from reaction ones, under the control of an application model. This pattern provides a structural scheme to enable the coordination, configuration and cooperation of distributed functionality within services platforms. Divides the information management related tasks from the ones that trigger actions, under the control of an application behaviour description. It provides the following structural scheme: A) Event, where tasks about the gathering and processing of context information are placed in; B) Control, that connects the context events with the actions to take; C) Action, that triggers the behaviour of the application.
- *Sense-Compute-Control (SCC) [97]*: Also known as Sensor-Controller-Actuator, this architectural pattern was tailored for applications that interact with the environment through sensors and actuators. Its architecture is divided in four layers [100]: I) Capture, which gathers the useful information from the environment; II) Refinement, where the information from the environment is treated; III) Control, where the decisions over the actions are taken; IV) Action, where the orders from the control layer are received and orders are executed. Although it presents a different layer for separating the acquisition of contextual information from its treatment, SCC resembles the Event-Control-Action (ECA) pattern, that presents solutions for recurring problems associated with managing context information and reacting upon context changes [101]. The main difference between these two patterns is the domain of application. While ECA fits better in environments where software-based changes are expected, SCC is conceived for systems that imply interactions among numerous heterogeneous devices, many of them directly interacting with their physical surroundings through sensors and actuators [102]. This pattern can be found in disparate application domains such as [100][102]: Ubiquitous computing, automotive & avionics, smart houses/offices/cities and industrial control systems among others.
- *Actions Pattern [98]*: It provides a structure of components that support designing and implementing of action-related concerns. Its structure divides action purposes from action implementations in order to better coordinate the composition of actions. Its component arrangement includes Action Resolvers, Action Providers and Action Implementors. The Action Resolver breaks compound actions into indivisible service units. Then, the Action Provider delegates them to the proper concrete action implementations (Action Implementor). An action might be performed independently or in parallel, while some actions depend on or trigger others.

The applicability and benefits of the presented architectural patterns can be observed in Table 4.

Pattern	Applicability	Benefits
Context Sources & Managers Hierarchy	• Context information processing • Decentralization	• Encapsulation, effective, flexible and decoupled distribution of context information management activities • Filtering unnecessary information
Blackboards	• Context information processing • Centralization • Complex problem solving	• Knowledge is reusable • Knowledge sources can work concurrently • Easy to add/remove knowledge
Event-Control-Action	• Trigger-action applications • Software-based actions	• Distribution of responsibilities • Dynamic deployment and development of applications • Extensible and flexible applications
Sense-Compute-Control	• Trigger-action applications • Large number of heterogeneous devices (sensors and actuators) • Static environments	• Distribution of responsibilities • Dynamic deployment and development of applications • Extensible and flexible applications • Allows the control of various devices
Actions Pattern	• Action management • Actions performed in parallel • Dependency between actions	• Avoids permanent binding between action and purpose • Enables different implementations at platform run-time • Actions may be changed or extended independently

Table 4: Benefits and applicability of architectural patterns for context-aware systems.

3.3.3. Middleware

Middleware is the most used structure to collect context information, support the deployment of sensors and hide heterogeneity. By separating how context is used from how it is acquired, it eases the development of a generic set of applications by reusing and customizing the necessary structure for context manipulation. Although many middleware approaches have been presented, it is difficult to achieve a universal middleware tool, applicable to any area, and capable of solving all the challenges involved in the context provision to applications, despite the many standpoints in the literature [9]. In this direction, approaches like UniversAAL[6], aim at creating an open platform and reference specification that makes technically feasible and economically viable to develop Ambient Assisted Living solutions. In order to better reuse their middleware, they provide a market in which developers will upload their applications to make them available to users. Besides, the ReAAL[7] initiative has rolled-out active and independent living applications on top of UniversAAL, and has tested them with users in real life. Middleware has been deeply analysed in the literature. For this reason, it is out of the scope of this survey to deeply analyse them. The reader can find broader comparisons and studies related to middleware in Perera et al. [9], Preuveneers and Novais [92], Kjaer [103], Baldauf et al. [96] or Henricksen et al. [104], among others.

[6] http://www.universaal.org/
[7] http://www.cip-reaal.eu/

3.3.4. Design patterns

A design pattern is a semi-structured description of an expert's method for solving a recurrent problem, which includes a description of the problem itself and the context in which the method is applicable, but does not include directives which bind the solution to unique circumstances [105]. As in other domains, design patters have also been suggested for context-aware computing. They can help designers to focus on what they want to implement without having to resolve recurrent issues. Usually, problems have a strong relationship with the platform where they are going to be executed, which makes their identification ad-hoc and difficult to reuse. We have classified the different patterns that can be obtained from the literature, according to the problem they intend to solve:

- *Ubicomp features:* Enable ubiquitous and pervasive computing features.
- *Fluid interactions:* Solve common problems that arise from providing a better interaction with the users.
- *Privacy:* Address issues related with the confidentiality of the user data.
- *Physical-virtual spaces:* Looks at how physical objects and spaces can be merged with the virtual.
- *Monitoring:* Enable to systematically observe the system itself and environmental conditions.
- *Adaptations:* To dynamically perform structural and behavioural changes in an adaptive system without leaving it in an erroneous or inconsistent state.
- *Decision making:* Mechanisms to solve problems related with taking decisions.

It is difficult to determine design patterns that can be universally reused. Instead, they help to solve some specific problems that might not be necessarily applicable to any C-AS. Besides, it must also be acknowledged that there is no widely recognized technique for finding appropriate design patterns from existing C-AS. Due to space restrictions, a further analysis of these patterns is not provided, but each of the patterns is related to its corresponding paper in table 5. More information about design patterns can be also found in [92] [106] [107].

Ubicomp features & Decision making	**Fluid Interactions**	**Privacy**
Ubicomp features	[109] [108] Follow-me displays	[108] Fair information practices
[108] Upfront Value Proposition	[109] [108] Context-sensitive I/O	[108] Respecting social organizations
[108] Personal Ubicomp	[110] Typified context element	[108] Building trust and credibility
[108] Ubicomp for groups	[108] Scale of interaction	[108] Reasonable level of control
[108] Ubicomp for places	[108] Sense-making of services and devices	[108] Appropriate privacy feedback
[108] Exploration and navigation guides	[108] Streamlining repetitive tasks	[108] Privacy sensitive architectures
[108] Enhanced emergency response	[108] Keeping users in control	[108] Partial identification
[108] Personal memory aids	[108] Serendipity in exploration	[108] Physical privacy zones
[108] Smart homes	[108] Active teaching	[108] Blurred personal data
[108] Augmented reality games	[108] Resolving ambiguity	[108] Limited access to personal data
[108] Streamlining business operations	[108] Ambient displays	[108] Invisible mode
[109] Global data proxies	[108] Pick and drop	[108] Limited data retention
Decision making	[109] Appropriate levels of attention and anticipation	[108] Notification on access of personal information
[107] Adaptation detector		[108] Privacy mirrors
[107] Case-based reasoning		[108] Keep personal data on personal devices
[107] Divide & conquer		
[107] Architecture-based		
[107] Trade-off based		

Monitoring	**Physical-Virtual Spaces**	**Adaptations**
[111] Flyweight	[109] [108] Physical-virtual associations	[111] Strategy
[111] Hybrid mediator-observer	[110] Active context element	[110] Rule-based adaptation
[111] Enactor	[108] Active Map	[110] Context wrapper
[111] Flexible context processing	[108] Tropical information	[107] Component insertion
[107] Sensor factory	[108] Successful experience capture	[107] Component removal
[107] Reflective monitoring	[108] User-created content	[107] Server reconfiguration
[107] Context-based routing	[108] Find a place	[107] Decentralized reconfiguration
	[108] Find a friend	
	[108] Notifier	

Table 5: Classification of design patterns used for context-aware systems development based on their applicability.

3.3.5. Design evaluation

The system complexity and hence the likely number of design errors, grows exponentially with the number of interacting system components. Although program testing can be a very effective way to show the presence of bugs, it is inadequate for showing their absence [112]. In these cases, verification techniques are used to explore some general properties about the behaviour of a program. Most of the verification done in C-AS is in the form of model checking, an approach to formal verification that proves whether if a model meets a given specification. Along with verification techniques, C-AS are usually evaluated by using simulations. In these experiments, the behaviour of a system is imitated in order to provide a preliminary understanding of its performance. The rest of the section discusses some representative samples of the state-of-the-art for evaluating the design of a C-AS.

3.3.5.1. Formal Verification. Formal verification techniques provide a safer development of systems in intelligent environments, what leads to increase their reliability [113]. Augusto et al. [114] show techniques as well as tools that can be used to model processes and interactions, detecting problems through simulation and verification in early stages of the development. On a further work, Augusto and Hornos [115], present a methodological guide which provides strategies and suggestions on how to model, simulate and verify these types of systems. Is divided in four stages: A) Informal modelling; B) Structural modelling; C) Behavioural modelling; D) Simulation and verification. The methodology is centred on a refinement strategy which starts identifying the core components of Intelligent Environments (sensors/actuators, actors, interfaces and communication mechanisms) and then working on successive models of increasing complexity. Although their methodology is tool-independent, they illustrate it using SPIN [116], a generic and open verification system that supports the design and verification of asynchronous process systems. Preuveneers and Berbers [117] also support a model checking approach in order to being able to verify the many possible configurations and contextual situation that a C-AS can be in. They discuss

the major benefits and weaknesses of the SPIN tool. D'Errico and Loreti [118] present a set of formal tools that allows specifying systems along with a model-checking algorithm to verify whether considered specification satisfy the expected properties. They introduce μKLAIM, based on a simplified version of a Kernel language for agent interaction and mobility [119], which is based on an assume-guarantee approach: A system is not considered as isolated, but in conjunction with assumptions on the environment behaviour where is executed. The system can be specified in: I) Process, accurately defined; II) Environment, more abstract and formalized by logical formulae. To specify properties of μKLAIM systems they use modal logic (MoMo) that allows describing interactions that the enclosing environments can have. Liu et al. [120] present AFChecker, a public available tool to improve user's fault detection and inspection experiences. It has three major components: 1) *Model checker* based on a technique for fault patterns and their automated identification [121]. Which derives a state transition model from a set of user-configured adaptation rules and verifies the model to detect five[8] common types of adaptation faults; 2) *Constraint inference engine*, that infers both deterministic and probabilistic constraints based on CHOCO[9] by analysing the propositional atoms in the user-configured adaptation rules; 3) *Fault Report Processor*, that processes the fault reports generated by its underlying model checker. The ranking of fault reports for user's inspection can be dynamic or static, depending on the interaction mode.

Approach	Applicability	Pros	Cons
• **Tool:** SPIN [116] • **Language:** PROMELA [122] • **Methodology:** MIRIE [115]	Intelligent Environments [113], Multi-agent systems [123], Environments with multiple devices and/or sensors	• Identify contexts that can give rise to conflicting actions • Supportive finding non-deterministic system behaviours • Provide counter-examples for unverifiable situations	• Natural explicit representation of time • Difficult to model external influences • Easy to overlook dependencies among context variables • In complex situations, many state spaces are difficult to process efficiently
• **Tool:** μKLAIM [118] • **Language:** KLAIM [119] MoMo [124] • **Methodology:** μKLAIM [118]	Code on Demand, Remote Evaluation, Mobile Agents, Distributed Systems	• Allows specifying systems by means of mixed specifications • Associativity and commutativity of parallel and non-deterministic operators	• Descriptions of the whole system are required to stablish system properties
• **Tool:** AFChecker [120] • **Language:** CHOCO[9] • **Methodology:** ×	User-configured rule-based adaptations	• Alleviates the false positive problem • Users can validate their own rules	• Only considers constraints on a binary basis • Might be too resource consuming for executing in a mobile device

Table 6: Comparative analysis on the advantages and disadvantages of the different approaches for validating context-aware systems through model checking.

[8]Non-deterministic adaptations, dead rule predicates, dead states (meaning that no rules can be satisfied in these states), adaptation traces and unreachable states.

[9]Open source Java library. http://choco-solver.org/

3.3.5.2. Simulation and test-case generation. Park et al. [125] present CASS, a simulation tool for smart-homes that is able to generate virtual people in order to perceive its movements and actions through sensors. The tool is programmed in Java, and it allows modifying and deleting sensors/devices according to the developers preferences. After, it can perceive simulated movements of virtual people, generating proper values for each sensor type. They also describe the system architecture and hierarchical rule structure model for smart-homes. Wang et al. [126], provide an approach for automating the generation of tests for context-aware pervasive applications. They provide an integrated solution to identify when context changes may be relevant, and a control mechanism to guide the execution of tests into potentially interesting contextual scenarios as defined by a coverage criterion that is context-cognizant. Their solution can be used to enhance other test suites of context-aware applications.

Bertran et al. [127] introduced DiaSuite, a tool suite for the development of sense-compute-control applications. Within their suite of tools, they present DiaSim [128], a parametrized simulator to ease the acquisition, testing and interfacing of a variety of software and hardware components. The simulator is parametrized to a high-level description of the target environment, written in their own specification language (DiaSpec). This description is used to generate both a programming framework to develop the simulation logic and an emulation layer to execute applications. Furthermore, the simulation can be rendered, allowing to visually monitor and debug the system. Their tool can be found as an Eclipse[10] plugin. Yu et al. [129] apply a bi-graphical reaction system to model the environment that interacts with the middleware and domain services in the development of C-AS. To model the data entities in the environment, they extend the bi-graphical sorting predicate logic and build a meta-model. Then, they create a model of the middleware using an extended finite state machine. By synchronizing the bi-graphical reaction system with the state machine, they can generate test cases to verify the interactions between the environment and the middleware. Finally, they show the reductions of the number of test cases by using a bi-graphical pattern-flow based testing on an airport example. Their tool is also in the form of an Eclipse[1]0 plugin.

Generally, authors recognize three main issues when simulating C-AS[127][129]: Modelling, source simulation and performance. First, it is difficult to determine what to model and in what granularity. Likewise, the model needs to be accurate enough to match such granularity. Second, some issues about the correctness of the stimulus producers may arise when either the logged data are replayed from actual sensors or a domain-specific modelling function is introduced. Emulated sensors must be programmed in such way, that for a given input, they produce the same output as its equivalent real sensor. Besides, merging the different intensities of simulated sensors requires

[10] Eclipse is an integrated development environment (IDE) from the open source community of tools, projects and collaborative working groups Eclipse. https://eclipse.org/

domain-specific knowledge. Finally, physical spaces may involve lots of services, accurate simulation models and rich simulation logics which can be resource consuming.

Approach	Features	Limitations	Tool Support
CASS [125]	• Simulate virtual people perceiving simulated movements in sensors • Able to detect rule conflicts	• Very ad-hoc • Only applicable to smart home development	✓
Automated generation of tests [126]	• Enhance existing test-suites • Identify when context changes might be relevant	• Static analysis tools are conservative • Infeasible drivers	×
DiaSim [128]	• Automatically generates an emulation layer to run the application code unchanged • Generation of a simulation framework to allow the development of the simulation logic	• The simulation logic has to be done by developers	✓
Bi-graphs & EFSM [129]	• Test cases are generated tracing the interactions between the bi-graphical model and the middleware • The number of test cases is reduced by using a bi-graphical pattern flow	• Assumes that middleware invokes only atomic services • Reaction rules are triggered in matches with agents and/or a middleware analysis result	✓

Table 7: Comparative analysis on the features and limitations of the different approaches for evaluating context-aware systems through simulations.

3.4. Implementation

After a good design and verified plan, there is a need to realize the implementation of the ideas into a tangible system. In this subsection we analyse the most common techniques for context information management and the most acknowledged programming paradigms that have been used for C-AS development.

3.4.1. Context information management techniques

There has been some research in what regards to context information management techniques [130] [131] [61] [132] [96]. Perera et al. [9], presented what we consider the most complete survey on it. They classified the context information life-cycle into: 1) Acquisition; 2) Modelling; 3) Reasoning; and 4) Dissemination. They analysed each of the techniques and compared their advantages and disadvantages. Following, a brief summary of the main techniques for this purpose.

3.4.1.1. Acquisition. For acquiring context, they discuss five factors that need to be considered when developing context-aware middleware solutions:

I) *Responsibility*: (a) *Pull*, the data is obtained from the sensors with a request; (b) *Push*, the sensor gives the data (periodically or instantly) to the software component that is responsible of obtaining it.

II) *Frequency*: (a) *Instant*, when events occur instantly (*e.g.*, Switching on a light or opening a door); (b) *Interval*, when events span a certain period.

III) *Source*: (a) Directly from sensor hardware; (b) Acquire through a middleware infrastructure or solution; (c) Acquire from context servers (*e.g.*, databases or web services).

IV) *Sensor types [132]*: (a) *Physical*, that generate sensor data by themselves; (b) *Virtual*, that do not necessary create sensor data by themselves and can retrieve data from many sources publishing it as sensor data; (c) *Logical,* that combine physical and virtual sensors to produce more meaningful information.

V) *Acquisition process*: (a) *Sense*, in which the data is sensed through sensors, including the data stored in databases; (b) *Derive*, in which the information is generated by performing computational operations on sensor data; (c) *Manually provided*, in which users provide context information manually via preferred setting options such as preferences.

3.4.1.2. Modelling and Representation Techniques. In order to implement models related to context, there is a need of platforms and techniques with the power to support the expression and handling needs of context information. Below, a brief introduction to the most commonly used techniques for context modelling [130]:

I) *Key-Value*: The simplest form of context models, involving a name and context value pairs. Used to model limited amount of data such as user preferences and application configurations. Contain mostly independent and non-related pieces of information, which are suitable for limited data transferring and any other less complex temporary modelling requirements.

II) *Markup Scheme*: Hierarchical data structures are formed using these models, consisting of mark-up tags, attributes and content. It can be the intermediate data organisation format as well as mode of data transfer over network. It can be used to decouple data structures used by two components in a system.

III) *Graphical*: Modelling of context using graphical notation as UML, Object-Role Modelling, and other DSLs. Ideal for long term and large volume of permanent data archival. Historic context can be stored in databases.

IV) *Object Oriented*: Take advantage of object-oriented concepts and techniques as encapsulation and inheritance. To represent context in programming code level. It allows context runtime manipulation.They work on a very short term, temporary and mostly stored in computer memory. Also, support data transfer over network.

V) *Logic Based*: Use facts, expressions and rules to define formal models. Different facts can be inferred separately and then used in existing rules to derive higher context knowledge. It is used for generating high-level context using low-level one, generating new knowledge. It is also used for modelling events and actions as well as for defining constraints and restrictions.

VI) *Ontology Based*: Can be used to describe taxonomies of concepts, including relationships. Besides, they allow different context reasoning techniques and inference rules. Rather than storing data on ontologies, data can be stored in appropriate data sources, while structure is provided by ontologies.

All techniques have their strong points and drawbacks, although ontologies are the most widely adopted approaches, still have some deficiencies that could be mitigated in hybrid approaches [61]. Although the representation and information retrieval in ontologies can be complex, they support semantic reasoning, expressive representations of context, have strong validation, are application independent, allow sharing, have strong support by standardizations and have fairly sophisticated tools available.

3.4.1.3. Reasoning. Once the context is modelled, there is a need of creating new knowledge and have a better understanding based on the currently sensed context. Techniques for this purpose can be divided into [9]:

I) *Supervised Learning*: Training examples are collected to label them according to the expected results. Finally, a function can generate the expected results using the training data. Techniques such as decision trees, Bayesian Networks, Artificial Neural Networks and Support Vector Machines are considered in this group.
II) *Unsupervised Learning*: Techniques that can find hidden structures in unlabelled data. Such as K-nearest neighbour, Kohonen Self Organizing Map (KSOM), Noise and outlier detection and Support Vector Machines.
III) *Rules*: One of the simplest, straightforward and popular reasoning methods. They usually have an IF-THEN-ELSE structure, but they can be based on simple mapping associations of IDs to entities (RFID) [38].
IV) *Fuzzy Logic*: Allows approximate reasoning instead of fixed one, extending the Boolean values form 0 or 1 to expressions that simulate closeness to a natural language. The confidence values represent degrees of membership rather than probability.
V) *Ontological*: Based on description logic, ontological reasoning is supported by OWL and RDF, rules as SWRL, are increasingly popular.
VI) *Probabilistic*: It allows decisions to be made based on probabilities attached to the facts related to the problem. These include techniques such as Dempster-Shafer, Hidden Markov Models and Naive Bayes.

3.4.1.4. Dissemination. Once the context information is ready, it has to be distributed to the consumers, and it is closely related to context acquisition. The distribution techniques are:

I) Query: The context consumer makes a request in certain manner so that they can obtain some specific results.
II) Subscription: The context consumer subscribes with the context management system. Then, this system will return the results periodically.

3.4.2. Programming paradigms

A programming paradigm is the structuring of thought that determines the foundation of a programming activity, influencing the structure and elements of programs. In this subsection, the mostly used programming paradigms for the C-AS development are briefly analysed. We have classified them into: A) Conventional; B) Emerging; C) Model Driven. The intention is not to have thorough research on them, but to highlight the most important bits on each of them. Due to space constraints, many papers have been omitted from each approach. A table comparing all the paradigms can be found in Table 8.

3.4.2.1. Conventional programming paradigms. In this subsection we briefly overview approaches based on programming paradigms that were originally conceived for the creation of conventional software systems, but that have been tried for the development of C-AS. We have divided it into four different approaches: A) Object-oriented; B) Aspect-oriented; C) Feature-oriented; and D) Service-oriented.

A) Object-oriented: Is the dominant programming paradigm for conventional software development. Considers the idea of building a software system by decomposing a problem into objects. These, abstract together behaviour and data into a single conceptual entity. Fortier et al. [133] introduce a programming and execution model supporting the development and execution of location-aware applications in mobile distributed systems. They show how to: A) Separate application concerns of C-AS to improve modularity; B) Ojectify context-aware services; Deal with contextual information; C) Model context; and D) Take advantage of transparent distribution mechanisms in mobile environments. Graff et al. [134] present an architecture for developing context-aware applications. They use and extend the dependency mechanism to connect different layers to avoid cluttering the application with rules or customization code.

B) Aspect-oriented: This paradigm [135] complements the object-oriented approach and was created for giving response to the programming problems for which neither procedural nor object-oriented programming techniques were sufficient to clearly capture important design decisions that a program must implement. It addresses the cases where behaviours have a difficult to define structure, because they are scattered across methods, classes, object hierarchies or even entire object models. For this purpose the approach uses *Aspects*, properties for which the implementation can not be cleanly encapsulated in a generalized procedure. The attractiveness for C-AS developers is that enables a system to adopt new features as it is created. Developers can dynamically modify the static object-oriented model to grow a system in order to meet new requirements. Tanter et al. [136] present an open framework for context-aware aspects to both restrict the scope of aspects according to the context and allow aspect definitions to access information associated to the context. Dantas et al. [137] show a comparative study on aspect-oriented pro-

	Object-oriented	Aspect-oriented	Feature-oriented	Software Product Lines	Service-oriented	Agent-oriented	Holoparadigm	Context-oriented	Model-driven	Domain Specific Language	Trigger-action programming
Building Unit	*Object:* Combination of variables, functions and structures	*Aspect:* Properties for which the implementation can not be cleanly encapsulated in a generalized procedure	*Feature:* Unit of functionality of a software system that: 1) Satisfies a requirement; 2) Represents a design decision; or 3) Provides a potential configuration option	*Product Line:* Satisfy different specific needs by using a common set of core assets in a prescribed way	*Service:* Software available to customers over a network	*Agent:* Entity whose state consists of mental properties (beliefs, capabilities, choices and commitments)	*Being:* Combination of interface, behaviour and history	Combination of: 1) Behavioural variations; 2) Layers; 3) Layer activation/deactivation mechanisms; and 4) Layer scope	*Model:* Simplified representation of a concept	*Domain Model:* Notation tailored to express the relevant concepts, features and topics that are related to a specific problem.	Conditional statements in the form of "if this then that"
General Applicability	• Encapsulating behaviour and data	• Encapsulating cross-cutting concerns (Homogeneous, dynamic and feature integration [140])	• Encapsulating cross-cutting concerns (Heterogeneous, Static and feature composition [140])	• Manage variability	• Integrate distributed systems	• Complex communication [141] • Entity cooperation, negotiation and competition • Autonomous behaviour [141] • Variable system purpose [141]	• Ubiquitous computing • Distributed programs	• Adapt the behaviour of entities dynamically [142]	• Deal with complexity • Guide the development	• Specialization of features for a particular domain	• Allowing unexperienced users to program the behaviour of different devices and applications
Advantages	• Modularity • Maintain-ability (Modifiability, Extensibility and Re-usability) • Mature approach	• Complementation of object-oriented • Better maintainability and code understanding	• Complementation of object-oriented • Better maintainability and code understanding	• Re-usability • Software quality control • Better requirements analysis • Reduce maintenance and testing costs	• Platform and Location independence • Maintainability (Modularity and Reuse) • Scalability	• Processing speed-up [143] • Reduced communication bandwidth [143] • Increased reliability [143]	• Not evaluated yet	• Enable software entities to adapt their behaviour dynamically [142] • Layers increase re-usability [142]	• Reduce risk • Better understanding among stakeholders • Reduce costs	• Productivity • Long term cost • Platform independence	• Offers greater motivation, control, ownership, creativity and quality to end-users [54] • Users are in control; users know their tasks best [51] • Releases developers burden •
Disadvantages	• Imperfect languages (Speed and Size) • Planning effort	• How to specify aspects • What composition and implementation mechanisms to provide • What implementation	• Architectural integrity • Superficial knowledge on layers	• Requires communication through the organization [144] • Lack of guidelines, techniques and tools for product line architecture design [144] • Expert knowledge of the application domain [144]	• Availability • Security • Investment cost • Response time and machine load • Service management complexity	• The benefits of this paradigm have not been demonstrated [145] • Still maturing area	• Not evaluated yet	• Current programming language implementations could have performance issues [146] • Do not support static changes • Still maturing area	• Specifying details • Introduces rigidity	• Short term cost • Training personnel • Too specific/generic models • Domain Knowledge required	• Users have to think about rules or find them on a store • Systems may become incompatible or introduce inconsistencies

Table 8: Comparative analysis on the advantages and disadvantages of the different programming paradigms used or created for context-aware system development.

gramming for C-AS, identifying CSAspectAJ as the most complete between the evaluated approaches, in what regards to synchronization issues, transparency, joint-point models, exception handling and implementation availability. Fuentes et al. [138] [139] presented an approach to design and implement aspect-oriented context-aware applications, run and test the design models, and show how these models map into an implementation.

C) *Feature-oriented:* Feature-Oriented Software Development (FOSD) is a paradigm for the construction, customization, and synthesis of large-scale software systems [147]. A *feature* is a unit of functionality of a software system that satisfies a requirement, represents a design decision, and provides a potential configuration option [147]. To this level of detail, feature-oriented programming resembles aspect-oriented programming. Both paradigms focus on a specific class of design and implementation of problems called *cross-cutting concerns*. These, are a design decision or issue whose implementation is scattered through the modules of code, violating the separation of concerns and modularity. Nevertheless, they are not competing approaches and can be used in combination to overcome individual limitations [140]. Ubayashi et al. [148] try to reduce the complexity of context-aware design by separating concerns. The demand of C-AS for static changes can benefit from the use of Software Product Line Engineering, reusing artefacts over a set of similar programs, called a Software Product Line (SPL) [149]. Both feature-oriented and aspect-oriented paradigms have been used along with SPL for C-AS development. Nevertheless, it has to be mentioned that , the approaches using SPL and the aspect-oriented paradigm are more popular. Following, we analyse more in depth the combination of them:

   - *Software Product Lines (SPL):* Fernandes et al. [150] [151] propose UbiFEX, an approach that supports feature analysis process for context-aware SPL and feature notation that provides context information representation as well as context rules specification. Parra et al. [152] create a composition of assets binding context adaptation to features for a context-aware Dynamic Software Product Line (DSPL), named CAPucine. In mobile computing, Marinho et al. [153] show a SPL for mobile and context-aware applications, along with the approach used to build it and a verification mechanism [154]. Kramer et al. [155] present an approach to support static and dynamic variability of a single code base of GUI documents within features, providing tool support. They also present a generic context acquisition engine for mobile devices [156]. This engine is used as a single customizable acquisition mechanism which can monitor, manage and disseminate context information to applications that are running on the same device. It also supports the composition of captured context events.

D) *Service-oriented:* The service-oriented approach to programming is based on the idea of composing applications by discovering and invoking network-available services to accomplish some task. In this paradigm, services are used as fundamental elements for developing applications. Kapitsaki et al. [157] survey methodologies and solutions for context-aware service engineering. They also acknowledge that the service engineering community lacks of a universally accepted basic design and development principles that can lead to a uniform approach to context-aware service development. Abeywickrama [158] claims for the need of solid software engineering methodologies needed for context ware development and execution. They present a software-engineering-based approach, using a model-driven architecture, aspect-oriented modelling and formal model checking.

3.4.2.2. Emerging programming paradigms. In this subsection we focus on new emerging programming paradigms that can be potentially used for the development of C-AS or were created specifically to build them.

A) *Agent-oriented:* This paradigm stems from a branch of Artificial Intelligence (AI) that attempts to combine distributed systems, AI and software engineering in a single discipline [159] [160]. It adopts the agent [161] abstraction for software development, which introduces the notion of mentality into the programming environment. Under this approach, the agent is used as the basic building block for creating software. An agent is an entity, whose state is viewed as consisting of mental components such as beliefs, capabilities, choices and commitments. They are usually considered as able to control their own behaviour in the furtherance of their own goals, being autonomous [162]. Different agents can be combined in a so called Multi-Agent System (MAS), in order to solve problems that are more difficult or impossible to achieve for an individual agent or system. The approach has been acknowledged as a promising in C-AS development by some authors [63] [163]. As an example, Murukannaiah and Singh [164] present Xipho, an agent oriented methodology that assists the developer in systematically modelling a context-aware personal agent (CPA) via cognitive constructs.

B) *Holoparadigm:* Victoria-Barbosa et al. [165] present holoparadigm, which integrates different programming paradigms in order to develop distributed/embedded systems. The paradigm is based on an abstraction called *Being*, which can be elementary or composed of other beings. An elementary being is an atomic being without composition levels. Is divided into: (1) Interface, describing the possible interactions among beings, (2) Behaviour, that contains actions composed, which implement the being's functionality and (3) History, a synchronized shared storage space in a being, which supports the communication and synchronization among the behaviour actions. On the other hand, a composed being may be formed from other beings that can be executed concurrently and shares

the history with its component beings. In order to coordinate the actions a model is used based on blackboard architecture. In further works [166] [167] they propose to apply a programming model specifically designed for the specification of context-aware applications, based on holoparadigm. It is intended to simplify the mobility management and the implementation of C-AS.

C) *Context-oriented:* Context Oriented Programming (COP) [142] is a technique to enable context-dependent computation. Is concerned with programming language constructs to represent and manipulate behavioural variations. COP tries to isolate the definitions from the business logic of application, conceptually separating context provisioning from the execution of the adaptable software. They identify four essential language properties to support COP [142]: 1) Means to specify behavioural variations; 2) Means to group variations into layers; 3) Dynamic activation and deactivation of layers based on context; 4) Means to explicitly and dynamically control the scope of layers. Salvaneschi et al. [106] give an overview of the COP techniques from the perspective of software engineering, recognising it as an apparently natural approach for this kind of systems. They acknowledge that supporting dynamic adaptation through proper language-level abstractions allows addressing the issues of adaptive software and avoid the decision logic for adaptive applications' behaviour to be scattered. Appeltauer et al. [146] present a comparison of presented context-oriented programming languages and acknowledge that they still have some performance penalties.

3.4.2.3. Model Driven Development. As compilers let programmers specify what the machine should do instead of how it should do it, Model-Driven Development (MDD) [168] aims to specify the system via high-level abstraction models that will be transformed into code. Models aim to reduce risk, helping to understand both a complex problem and its potential solutions before undertaking the expense and effort of a full implementation [169]. Sheng and Benatallah [170] presented ContextUML, a modelling language based on the Unified Modelling Language (UML) [171] for the model-driven development of context-aware web services. Serral et al. [172] [173] introduce a model-driven development method for context-aware pervasive systems. It applies the Model-Driven Architecture (MDA) [168] and Software Factories (SF), along with the PervML modelling language and the SOUPA ontology. Tesoriero et al. [174] presented CAUCE, a methodology based on MDA [168], to provide a model-driven development of applications for Ubiquitous Computing environments. It is also worthy to be mentioned that there are some Domain Specific Languages (DSL) for the development of context-aware software systems [175] [176]. Recently, a domain specific language called Trigger-action programming [47] [45] [46] is gaining popularity. Following, we briefly explain it:

- *Trigger-action programming:* Is a programming model based on the End-User Development paradigm [44], where average users can manually customize a service according to their preferences, likes and expectations [46]. By reducing the complexity of programming, expressing the system behaviour becomes accessible to end-users. These, only need to handle simplified if-then programming rules that match a trigger with an action. Is starting to emerge in areas such as smart-homes/buildings [177] or smartphones [178]. Services and applications such as IFTTT [11] or Tasker [12] let end-users create rules with sensors/devices that they already have and use in their daily life. A recent study [45] has found that this approach can express the most desired behaviours in order to personalize smart-home devices. Through a usability test conducted to 226 participants, they encountered that users without experience can learn to create programs containing multiple triggers or actions obtained by extending the IFTTT language, that can express only one trigger and one action.

3.5. Deployment and Maintenance

Once the system is implemented, a typical life-cycle does not end. It is followed by evaluation and maintenance phases. Also, techniques such as documentation, training and support are highly recommended, as they help future maintenance and enhancement, as well as user acceptance. This subsection analyses the evaluation and maintenance techniques for a C-AS specialized development. Maintenance is the modification of a software product after its delivery in order to correct faults, improve performance or other attributes[13]. C-AS require handling change faster and cheaper than conventional approaches. An initial design tends to become outdated or insufficient fairly quickly because of changing requirements [44]. Despite the evolutionary nature of C-AS, it is difficult to find in the literature approaches exclusively focused on improving the maintenance of C-AS. In the classical software engineering paradigm, there are four core maintenance activities [179]: (1) Adaptive; (2) Perfective; (3) Corrective; and (4) Preventive.

3.5.0.4. Corrective. It is involved with fixing errors, faults or bugs in the system to restore. A bug is a defect that causes the system not to behave in the expected way. Debugging is the methodical process of finding a reducing the number of software and hardware defects in order to make the system behave in the expected way. It gets difficult to find bugs when it comes to classical programming, so in C-AS, where information is more complex to handle, it gets even more complicated. It has to be acknowledged that there is still very little research in specialized debugging methods for C-AS. Moos et al. [180] propose the use of intelligibility to help users debugging why the system is not working. In their approach, debugging for C-AS is introduced as a mean to assist the users in discovering the cause of

[11] IFTTT (If This Then That): Is a web-based service that allows users to create chains of simple conditional statements, triggered based on changes to other devices or web services (Facebook,Gmail,Calendar). https://ifttt.com/

[12] Tasker: An android application for performing tasks based on contexts (application,time,date,location,event,gesture) defined in user profiles or in clickable or timer home screen widgets. http://tasker.dinglisch.net/

[13] ISO/IEC 14764:2006

the failure. In order to achieve this, they propose to include an information exchange approach from "explanatory debugging".

3.5.0.5. Preventive. It tries to prevent problems with the system before they occur, anticipating adaptive maintenance needs before users experience problems. Failure handling issues are the most concerned theme of research for C-AS within maintenance. Chetan et al. [71] classify the possible failures into: (1) Device failures, due to the different kind of devices that conform a pervasive system; (2) Application failures, that include application crashes due to bugs, operating system errors, not handled exceptions, and faulty usage; (3) Network failures, due to the different connection channels that devices can have; (4) Service failures, as service crashes due to bugs and operating system errors, faulty operation of services, wrong inferring and lossy delivery of events. Kulkarni and Tripathi [181] present a framework for programming robust context applications. They use a recovery model that consists of mechanisms for asynchronous event handling and synchronous exception handling. It integrates event handling at the object level with exception handling at the role level to build robust role-based context-aware applications. The exception interface for roles provides the ability for users to handle exceptions. In order to complement their application-level recovery mechanisms, authors suggest to use techniques such as replicating the trusted servers and running the various managers in a primary backup. The techniques for failure detection can be classified into [71]:

- *Surrogate Application/Device Usage*: Upon failure, the process is restarted and restored from a stable storage device.
- *Alternate Notification mechanisms*: The system notifies the personnel trough different devices. If the system discovers that a notification device has failed, it should reroute the message trough a different notification channel.
- *Handling errors in sensing and inferring context:* Detecting errors happened during the sensing and inferring phases of context information. This could be done by employing redundancy (multiple sensors that sense the same), so that the results can be compared. Another technique could be to let users identify any errors that might experience.
- *N-Version approach*: Executing in N different implementations the same task and giving the correct answer to an arbitrator.
- *Fault notification mechanisms*: Notify the errors in the devices that the user is using. This creates a dependency graph that could span numerous applications, services and devices.

3.5.0.6. Adaptive & Perfective. Adaptive maintenance is involved with adapting the system to the ever changing hardware and software developments. The adaptiveness concern present in C-AS literature is more related with the behavioural changes that the context triggers, more than the platforms in which the system will be executing. On the other hand, perfective maintenance is concerned with the improvement of the system features. To the extent of our knowledge, there is very little adaptive or perfective maintenance techniques for the development of C-AS.

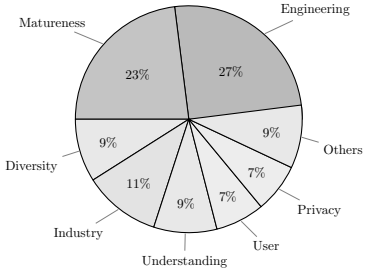

Figure 2: Pie chart showing the response rate in each of the categories.

4. Methodologies for context-aware systems development

The previous section analyses different techniques that have been used for developing C-AS. These, are typically focused on addressing a specific problem in the development and are generally independent from each other. On this section, we focus on the unification of different techniques into methodologies that can be used for C-AS development. We assess the needs of a methodology specifically conceived for this purpose through a questionnaire done to 750 researchers that have conducted some research related to C-AS development. First, we include experts opinion in order to clarify why there is no commonly accepted methodology for this purpose. Second, we identify which features would a methodology require to have better acceptance in the community. Finally, we study existing efforts in creating a unified methodology for C-AS development. For this, we study the coverage of actual methodologies for the most common development stages and for their desirable features according to the questionnaire.

4.1. Assessing the needs of a methodology

In an open question of our questionnaire, contestants were asked about the main reasons for not having a commonly accepted methodology or tool for developing marketable C-AS. The responses were classified in eight different categories: Engineering, Matureness, Diversity, Industry, Understanding, User, Privacy and Others. The pie chart from Figure 2, shows the response rate obtained in each of the categories.

a) *Engineering*: There are lack of standards for representing information, models and general-purpose support. Better managerial support should be provided once C-AS are rolled out along with proper documentation. These systems must integrate other sub-systems (that sometimes use emergent ever-changing technologies). Interoperability issues were recognized as well as the absence of common middleware solutions to ease its development. Besides, the diversity of hardware-software requirements, that tradeoff with each other and the absence of common vocabulary/concepts when developing C-AS has been acknowledged. The difficulty to adequate a prototype to a real sys-

tem has not been evaluated. The research field has a bigger focus in the deliverables more than in the engineering process. Finally, software development companies believe that the application of formal methods in the early stages of a project delays them.

b) *Matureness*: The immatureness of the field was acknowledged, due to the technology: Expensive, invasive-size, not too powerful/useful or that depends on other technologies that are still evolving. Also, the infrastructure is either still very expensive or it has not been developed for the public yet.

c) *Diversity*: Survey respondents also stated that there are many alternatives (SW Architectures, algorithms, methods, techniques, etc.), that can be required in a multiple type of developments (from operating systems to home automation), apart from the diversity of possible scenarios. One of the participants believes that context should be approached in different ways and another that the problem is that "*different developers/researchers focus on different aspects*".

d) *Industry*: There is a need for the industry to invest behind the development of these systems. Some even acknowledge that the reasons why companies do not invest money in C-AS are that: "*There is no clear business for "context-something" applications, users don't care, they have it already*" or that "*Daily life environments not being equipped with appropriate seamlessly integrated devices for delivering contextualized application's functions*".

e) *Understanding*: There is no shared understanding of context and systems get the term wrong. One of the participants highlighted that there is no common vocabulary and concepts for C-AS.

f) *User*: The user is a factor that influences the lack of acceptance. Participants report that the user opinions are not taken into account neither during the development nor while the system is executing. They also believe that users are not confident with C-AS.

g) *Privacy*: Is one of the reasons behind the absence of acquisition of tools/methodologies. Mainly because the user does not feel comfortable with "*a machine knowing too much about humans*". They also recognise the lack of full control about the collected data. One of the contributors to the poll, states that user privacy should be taken into account from the first stages of the design.

h) *Others*: A couple of experts referred to intelligibility and the control about the information of the user and the activities carried out in the environment. Others proposed that there was no union of communities that study the field and there is no reuse of knowledge between researchers/companies. Finally, one of the survey respondents believes that presented C-AS do not work in perfect (or nearly perfect) real-time environments.

4.2. Desirable features for a methodology

Participants were also asked to evaluate how important they considered some features in the development of C-AS. From 0 to 5, where 0 is the lowest in importance and 5 the highest. The participants were also asked to suggest features they would include in a methodology that were not considered in the previous questionnaire. The answers were similar to the proposed features. Results can be graphically observed in Figure 3. The choices given where:

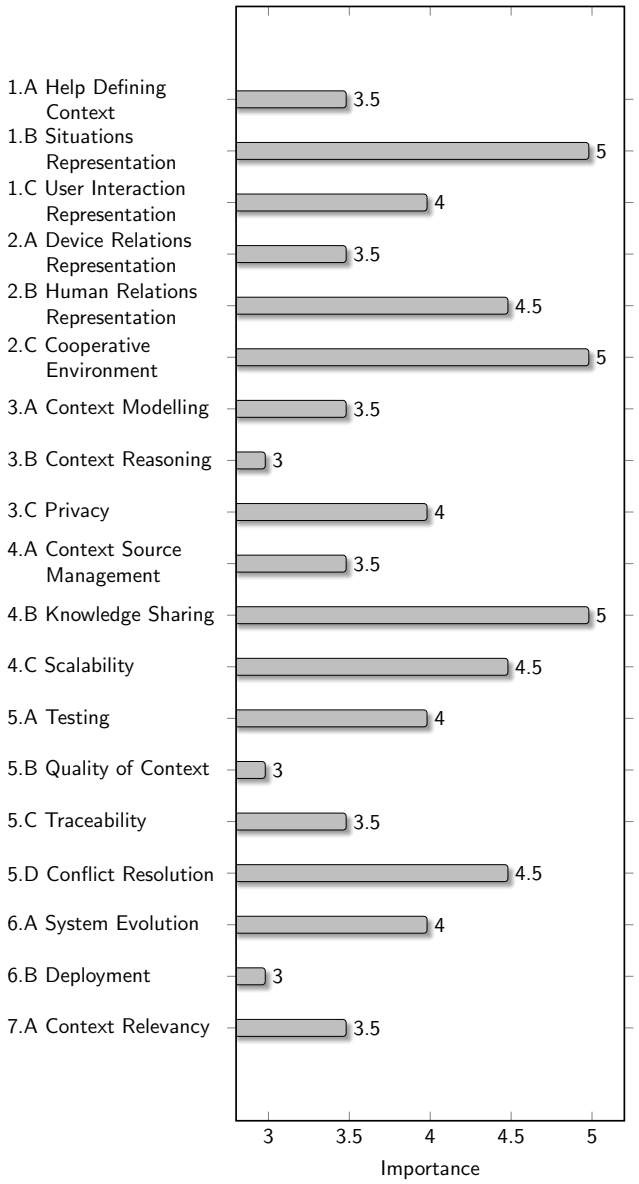

Figure 3: Bar graph showing the importance that contestants would give to including certain features in a context-aware development process.

1.A Help to Define Context: The support to understand the context notion within the boundaries of the system to be developed. For example, coining vocabulary to define the system features.

1.B Situations Representation: The ability to represent situations in which the system is intended to act in a certain way in order to better understand them.

1.C User Interaction Representation: To be able to represent and model the interactions between the system and the

users.

2.A *Device Relations Representation*: To represent and model the relations between devices.

2.B *Human Relations Representation*: To allow the representation and modelling of human relations and interactions.

2.C *Cooperative Environment*: To allow the combination of different environments in order to represent the details that would enable them to work together.

3.A *Context Modelling*: The ability to model the context information, for example using ontologies.

3.B *Context Reasoning*: To model the reasoning of the context information in order to choose the information that should infer or the actions that it should take.

3.C *Privacy*: Enable the secure use of the information relative to users, so that is not interfered by other people or organizations.

4.A *Context Source Management*: It explicitly specifies how the context data will be obtained.

4.B *Knowledge Sharing*: It allows defining how will the system distribute the knowledge within its own boundaries and outside them.

4.C *Scalability*: Supports the system to handle a growing amount of work effectively, or enables the system expansion/reduction to accommodate that growth/decrease.

5.A *Testing*: Ensures that the system meets its requirements.

5.B *Quality of Context*: It provides a good quality of precision, probability of correctness, trustworthiness, resolution and contemporaneity of context information [69].

5.C *Traceability*: Allows tracking a given set or type of information to a given degree.

5.D *Conflict Resolution*: Enable the conflict resolution of the C-AS, considering it as the process that enables a system to provide its safety-critical functionalities by recovering from errors and faults and preventing the system failure.

6.A *System Evolution*: Supports evolution and maintenance of the system.

6.B *Deployment*: Eases the system deployment.

7.A *Context Relevancy*: It allows defining which contexts are relevant depending on the situation.

4.3. Existing methodologies and tools

Following, a brief description of the main existing methodologies that are related to the C-AS development is provided. It has to be mentioned that methodologies published in conferences are often only theoretical and do not present tool support. For this reason, and due to space restrictions, the methodologies, frameworks and tools here presented are only the ones published on journals.

Context Toolkit [49]: Was one of the first efforts to facilitate the development and deployment of context-aware applications by providing a framework to support it. It provides abstractions to separate the details of how things are done from actually doing them: (1) Context Widget, to separate the details of sensing context from actually using it; (2) Context Interpreter, to reason sensor data using different reasoning techniques; (3) Context aggregator (Server), to collect multiple pieces of context information that are related into a common repository; (4) Enactors, that serve as application units that acquire and take actions based upon context. Widgets, servers, interpreters and enactors are allowed to run on different computers, communicating over a network. This toolkit has been further extended by other authors [38] [47].

ISAMadapt [182]: Provides an integrated environment aimed at building general-purpose pervasive applications. It works on the basis of four main abstractions: Context, adapter adaptation commands, and adaptive behaviour management policies. They focus on supporting the follow-me semantics for building generic applications for building pervasive applications, investigating how context-awareness can be expressed at the programming language level. They offer an integrated software infrastructure both to design pervasive applications and to manage a pervasive environment at global scale.

Context Modelling Language (CML) [183]: Was created as a tool to assist designers exploring and specifying the context requirements of context-aware applications. They propose a set of conceptual models to support the software engineering process, including context modelling techniques, a preference model for requirements representation and two programming models. Along with it, they present an engineering process supported by a software infrastructure. The infrastructure is divided in seven phases: (1) Context Gathering, that allows the use of context interpreters and aggregators; (2) Context reception layer; (4) Context management layer; (5) Query layer;(6) Adaptation layer and (7) Application layer. Their work introduced improved opportunities for tool support into the software engineering process.

The MUSIC[14] *Project:* Methodology to facilitate the development of adaptive applications in open, heterogeneous Ubiquitous Computing environments. The methodology includes tool support and an adaptation middleware and is based on the separation of concerns between the business logic, context-awareness and adaptation. Design and implementation of context-aware adaptive applications is done via model-driven development. They provide a software development framework for the automation of the adaptation of the software at run-time, including: (I) A modelling language; (II) Generic and reusable middleware components that automate text monitoring & management and adaptation; (III) Tools to support the development: such as design models, transformation, deployment, testing and validation ones.

OPEN: OPEN [184] is an ontology-based cooperative programming framework for the rapid prototyping, sharing, and personalization of context-aware applications for

[14]MUSIC [32] [33]: Component-based planning framework that optimizes context-aware variations, partially funded by the European Commission under research grant IST-035166 lasting from October 2006 to March 2010.

No.	Name	Year	Reference	Requirements Elicitation Technique	Middleware	Verification Technique	Maintenance Technique	Requirements Elicitation Tool	Analysis and Design Tool	Development Tool	Evaluation Tool	Maintenance Tool	Situations Representation	Cooperative Environments	Knowledge Sharing	Human Relations Representation	Scalability	Conflict Resolution
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)	(17)	(18)
(a)	Context Toolkit	2001	[49]	-	✓	-	-	-	✓	✓	-	-	✓	*	✓	*	✓	-
(b)	ISAMadapt	2002	[182]	-	✓	-	-	-	✓	✓	-	-	✓	*	-	*	✓	-
(c)	CML	2006	[183]	*	✓	WB,BB	-	-	✓	✓	✓	-	✓	*	✓	*	*	-
(d)	CAMUS	2007	[185]	-	-	-	-	-	✓	✓	-	-	✓	*	*	*	*	-
(e)	MUSIC	2012	[32]	*	✓	Sim.	*	-	✓	✓	✓	-	✓	*	*	*	✓	-
(f)	OPEN	2011	[184]	-	-	-	*	-	✓	✓	-	-	✓	*	*	*	✓	-
(g)	CA-PSCF	2010	[186]	-	✓	MD	*	-	✓	✓	-	-	*	*	*	*	✓	-
(h)	-	2010	[187]	So	✓	-	-	-	✓	✓	-	-	*	*	*	✓	✓	-
(i)	MIRIE	2013	[115]	-	-	Frm.	*	-	✓	-	✓	-	✓	*	-	*	✓	-
(j)	PerDe	2014	[188]	-	✓	Sim.	*	-	✓	✓	-	-	✓	*	-	✓	✓	-
(k)	DiaSuite	2014	[127]	-	✓	Sim.	Perf.,*	-	✓	✓	✓	✓	✓	*	-	*	✓	✓

Table 9: Comparison of existing methodologies to develop C-AS.

General Symbols for Techniques: "✓" = Uses a technique/Has tool support, "-" = Does not use a technique/ Does not have tool support; "*" = Mentions a technique but is not explained in detail; **(4) Requirements Elicitation Technique:** "So" = Social sciences based techniques; **(6) Verification Technique:** "WB" = White box verification, "BB" = Black box, "Sim." = Simulation based, "M.D." = Model driven, "Frm." = Formal **(7) Maintenance Technique:** "Perf."= Perfective;

	R.E. Context-Aware Specialised Techniques	R.E. Traditional Techniques	Architecture Patterns	Design Patterns	Middle-ware	Verifi-cation	Programming Paradigms	Context Information Management	Deployment & Mainte-nance Techniques
Situations Representation	Visualization	Scenario Based	-	-	*	*	*	Modelling, Reasoning	-
Cooperative Environments	-	-	-	*	*	*	*	-	
Knowledge Sharing	-	-	-	*	*	*	Holoparadigm	Dissemination, Acquisition	-
Human Relations Representation	Social Science Based	Ethnography Based, Observation	-	-	*	*	-	-	-
Scalability	-	-	ECA, CS&MH, SCC, Black-boards	*	*	*	FeO&SPL	*	Adaptive
Conflict Resolution	-	-	*	*	*	*	*	*	Preventive

ECA = Event-Control-Action, **CS&MH** = Context Source and Managers Hierarchy, **SCC** = Sense-Compute-Control;
FeO&SPL = Feature Oriented and Software Product Lines; **"-"** = There is no technique ; **"*"** = Not comparable within the scope of this survey.

Table 10: Comparison of the features that a methodology for C-AS development should support related to the state-of-the-art techniques.

users with diverse technical skills. To meet diverse developer requirements in the development and customization of context-aware applications, it implements three programming modes with diverse complexity: (1) Incremental mode, for high-level users, which supports the creation of new context-aware applications; (2) Composition mode, a programming mode for middle-level users; and (3) Parametrization mode, for low-level users, to enable them customize existing applications.

CA-PSCF [186]: is a model-driven approach to facilitate the creation of a context modelling framework that aims to simplify the design and implementation of pervasive services. It uses model-driven development to provide a systematic methodology that facilitates the generation of modelling frameworks and supports the overall service creation process. The process is as follows: (1) The oAW editor is used to define a code template, which conforms to a context meta-model; (2) With the context editor a context model is defined and validated; (3) With the context model and the code template, the work-flow execution engine generates service code. Optionally, models can be transformed(4): Source domain models are transformed to target domain models in a different domain language.

Human-Centred Computing Methodology for Cooperative Ambient Intelligence: Gross [187] introduces a cooperative Ambient Intelligence methodology, elaborated on existing approaches for organising software engineering and user-centred design processes. It suggests a new human-centred computing methodology for this aim. He acknowledges that many research issues remain for each phase in terms of: (1) Methods to be applied; (2) The adaptation of the method concerning the characteristics of the targeted technological innovation; and (3) The properties of the results of each phase. The life-cycle is divided into: (A) Identifying the need for cooperative Ambient Intelligence; (B) Understand the situation of use; (C) Specify the user, ambient and cooperation requirements; (D) Produce software and hardware design solutions; (E) Produce embedded interaction and adaptation design solutions; (F) Perform software and hardware evaluation; (G) Perform overall evaluation in living laboratory; (H) Reach specified user, ambient and cooperation satisfaction.

MIRIE [115]: The Methodology for Improving the Reliability of Intelligent Environments, has been created to guide and inform the development of more reliable Intelligent Environments. The methodology is centred on a refinement strategy which starts identifying the core components of the Intelligent Environment (sensors/actuators, actors, interfaces and communication mechanisms) and then working through successive models of increasing complexity. The methodology is illustrated with SPIN and using the PROMELA modelling language but it is actually tool-independent. In [115] a detailed explanation on the use of formal verification to support the development of Intelligent Environments is given. Whilst other approaches mention the possibility of using verification and give a few examples the above publication provides a step by step and detailed explanation of how developers can benefit from formal methods at a pragmatic and practical level. The methodology has been applied to several Intelligent Environments, one of which is a real Ambient Assisted Living system which is explained in the article as the main case study.

PerDe [188]: Is a development environment that orients the designs of Pervasive Computing applications towards the user's needs. It provides a domain-specific design language and a set of graphical tool-kits covering some development life-cycle stages for this kind of applications. It provides means to structure the application and allows the developer to build an application: (1) Focusing on the human situations evolved to the physical surrounding; (2) Characterizing the application along with attributes of a new application model by representing the intentions as the task requirements, service specification and instantiation of services in devices abstractly; (3) Specifying a programming structure that trades-off between the function requirement and the device capability; (4) Rapid prototyping.

DiaSuite [127]: Is a tool-based development methodology that uses a software design approach to drive the development process in the domain of Sense/Compute/Control (SCC) applications. It provides a design language called DiaSpec, dedicated to the SCC paradigm, based on two layers: *Taxonomy* and *application design*. The application specification can be represented by its data flow using an oriented graph. In addition, DiaSuite has a compiler that produces a dedicated Java programming framework, guiding the programmer to implement the various parts of the software system (Entities, context operators and control operators). The suite also includes a 2D-graphical renderer, for simulation proposes. The deployment can be done either in distributed execution platform or local.

4.4. Coverage of desirable features

The coverage of the most desirable features of our questionnaire compared to the existing methodologies can be observed in Table 9. We have considered to leave a deeper comparison out of this survey, as it is not possible to measure them in a useful way, due to their disparate aims and themes. Instead, we analyse the state-of-art techniques that could be used for creating the most desirable features of a methodology according to our questionnaire.

Situations Representation: During the requirement elicitation stage, techniques such as Executable Use Cases [189] or the tool presented by Perez and Valderas [190] can be used in order to represent situations that will help to gather context-related requirements. There is not much presence of situations representation during the rest of the most common stages of development. Although specific middleware infrastructures might have features for representing situations during this phase. All context information modelling and reasoning techniques need to enable

the situation representation, but there is no support for understanding the situations and the contexts that they are going to be represented, stemming from the requirements.

Cooperative Environments: The cooperation between environments is a technique that stems from Ubiquitous Computing. The support in what regards to techniques for the cooperation between environments in context-awareness is very little, there is more support for this in other fields like Systems Engineering or Distributed Systems, which are out of the scope of this survey.

Knowledge Sharing: Is not taken into account during the early stages of the development. The techniques used for these are Queries and Subscribers as explained in Section 3.4.1.4. Although the deep analysis of programming paradigms is out of the scope of this survey it has to be mentioned that Holoparadigm, enables the knowledge sharing at programming level.

Human Relations Representation: As discussed in section 2.1, the human relationships and interactions are explained using social sciences, which stem from a phenomenological philosophical tradition. Bauer et al. acknowledge that during the C-AS system development, in its life-cycle, starts from a more highlighted phenomenological perspective and proceeds slowly transforming into a more positivist one. We can observe that human relations representation have more presence in early stages of the development, and lose support as the development process continues. Using behavioural studies are helpful to capture more completely human relationships and interactions into requirements of the system as shown by Kjaer [81] and Fuentes et al. [191]. Ethnography based and observation techniques can also be used for this purpose.

Scalability: The majority of the architecture patterns that are specific for C-AS development enable the scalability of the system. From these the highlighted ones are Event-Condition-Action, Context Source and Managers Hierarchy, Sense-Compute-Control and Blackboards. Nevertheless, it has to be mentioned that each of them enable different types of scalability and that a further analysis is out of the boundaries of this survey. Although the intention is not to have a thoughtful comparison of programming paradigms, Feature Oriented one is prominent. Especially when it tends to Software Product Lines. Finally, the classical adaptive maintenance techniques will enable also the scalability. It is very difficult to measure scalability in Design Patterns, Middleware and Verification.

Conflict Resolution: There is very little support during the whole life-cycle of context-aware development for conflict resolution. Preventive maintenance can help to enable this feature. Also, the framework presented by Kulkarni and Tripathi [181], as well as other techniques like Surrogate Application/Device Usage, Alternate notification mechanisms, Handling errors in sensing and inferring context, N-Version Approach and Fault Toleration Mechanisms.

The results of the comparison show quite weak support in the state-of-the-art techniques for the most desirable features of a methodology, as it can be observed in Table 10.

4.5. Coverage of development stages

Generally, the engineering techniques are more concerned with the design and information management of C-AS, there is less attention focused on other stages the development process. The following subsection analyses the techniques and tool support of the methodologies through these stages.

4.5.1. Techniques

In general, there are no techniques for the early and latest stages of the development process. The human-centred computing methodology [187] mentions that ethnomethodologically informed ethnography can be applied for understanding the situation of use, but they do not deeply explain how the technique should be applied. Also, MUSIC [32] and CML [183] mention requirements, but do not explain into detail the techniques. DiaSuite [127] offers perfective maintenance for the system, but does not support adaptive, preventive or corrective ones. The most used verification technique is simulation, but generally the verification support is not very strong. Finally, it has to be mentioned that most of the methodologies offer middleware support.

4.5.2. Tool support

There is strong tool support for the design and development of C-AS. Nevertheless, they do not offer support for other stages. No methodology enables the requirement elicitation with a tool, and only DiaSuite [127] allows the maintenance with one. MIRIE [115] offers a strong formal verification support, while MUSIC [32], PerDe [188] and DiaSuite [127] offer verification based on simulation. It has to be acknowledged that DiaSuite [127] is one of the most complete methodologies, regarding to these factors and within the scope of this survey.

5. Conclusion

This survey has analysed the concept of context and the reasons behind the lack of agreement on its definition. Taking into account the limitations of C-AS, we have characterized their interaction types and features. Then, we analyse the main challenges in its development. The literature review shows various techniques and methods that have been modified from the conventional development to better fit the needs of C-AS creation. We provide an analysis of these techniques during different stages of a development life cycle. We acknowledge that they only target certain aspects of the engineering process, sometimes only solving specific problems of a system. In order to go one step further, we focus on methodologies. The results show little support for the most common stages of a development process. We have carried out a questionnaire to understand the main reasons for the lack of adoption of these methodologies, as well as to identify the features of a new methodology that could have more acceptance in the

community. Our study shows that existing methodologies lack some features that will make them be accepted by the community.

Contextual awareness is an attractive feature that has encouraged research in the latest decades. Embedding such feature into a system implies challenges that are to be solved yet. There is a notable difference between developing a conventional system and a context-aware one. Although there is lot of research related to the development of this kind of systems, this is focused on solving particular issues. The evidence presented in this work supports the need of a more holistic and unified approach for the development of C-AS. Also it should be different from the classical software engineering approach for creating these systems. Context-Aware Systems Engineering (C-ASE) is the study and application of engineering techniques to any activity involved in the creation of C-AS, including [192] its analysis, design, assessment, implementation, test maintenance and re-engineering, among others. Below, we present further research directions for C-ASE:

- *Design Principles and Human Computer Interaction*: There is a need to better understand the potential synergies between phenomenology and positivism philosophical paradigms in what regards to C-AS engineering. What are the specific limitations of a C-AS? When can developers use an autonomous approach? When is it more recommendable to use a passive one? Not only a further philosophical research would give a better understanding on when to use a certain interaction type. It would also shed light on the conceptualization of context.
- *Diversity*: The diversity among systems that could exhibit contextual awareness makes difficult the creation of generalized methods that can be applied to all of them. The research in C-ASE should also focus on the differences among the contextual awareness in different areas of application.
- *Information management*: The technology for handling contextual information is the most matured in the field of context-aware computing. Nevertheless, it has still its challenges to overcome, as explained in Section 3.1.2. Further research is required in this direction.
- *Requirements elicitation*: We acknowledge that requirement elicitation techniques in context-awareness are still maturing. The research directions point towards the benefits from merging with social science based approaches. There is also a need to measure to what extent standardized methods such as SysML or UML cover the needs of C-AS for this stage of its development.
- *Architectural patterns*: What are the most common design problems of C-AS architecture? Can these be categorized? Can a tailored solution for each of the problems be proposed?
- *Programming paradigms*: There is a demand to further study the synergies between the different programming paradigms. Which paradigm is better for implementing which features of context-awareness? When does a developer know that there is a need to use one or another? Which is the best programming language for C-AS development?
- *Maintenance and deployment*: Once the system is implemented, which are the best ways of detecting an error? The maintenance and deployment has also several challenges that need to be further investigated.
- *Methodological support*: There is a need for a methodology that guides the development of a context-aware system through all its stages. Standardized methods and tool support for methodologies are future research objectives to take into account.

Acknowledgements

The authors would like to acknowledge the anonymous reviewers for the comments that have helped to improve the final result of this survey.

References

[1] M. Weiser, The computer for the 21st century, Scientific american 265 (3) (1991) 94–104.
[2] T. Erickson, Some problems with the notion of context-aware computing, Communications of the ACM 45 (2) (2002) 102–104.
[3] A. K. Dey, G. D. Abowd, Towards a better understanding of context and context-awareness, in: In HUC 99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing, Springer-Verlag, 1999, pp. 304–307.
[4] M. Bazire, P. Brézillon, Understanding context before using it, in: Modeling and using context, Springer, 2005, pp. 29–40.
[5] P. Dourish, What we talk about when we talk about context, Personal and ubiquitous computing 8 (1) (2004) 19–30.
[6] A. Zimmermann, A. Lorenz, R. Oppermann, An operational definition of context, in: Modeling and using context, Springer, 2007, pp. 558–571.
[7] S. Jumisko-Pyykkö, T. Vainio, Framing the context of use for mobile hci, International Journal of Mobile-Human-Computer-Interaction (IJMHCI) 3 (4) (2010) 1–28.
[8] Z. Alshaikh, C. Boughton, Notes on synthesis of context between engineering and social science, in: Modeling and Using Context, Springer, 2013, pp. 157–170.
[9] C. Perera, A. Zaslavsky, P. Christen, D. Georgakopoulos, Context aware computing for the internet of things: A survey, Communications Surveys & Tutorials, IEEE 16 (1) (2014) 414–454.
[10] A. K. Dey, Understanding and using context, Personal and Ubiquitous Computing 5 (2001) 4–7.
[11] K. Henricksen, A framework for context-aware pervasive computing applications, Ph.D. thesis, University of Queensland (2003).
[12] A. H. Van Bunningen, L. Feng, P. M. Apers, Context for ubiquitous data management, in: Ubiquitous Data Management, 2005. UDM 2005. International Workshop on, IEEE, 2005, pp. 17–24.
[13] B. Schilit, N. Adams, R. Want, Context-aware computing applications, in: Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on, IEEE, 1994, pp. 85–90.
[14] `https://www.google.co.uk/landing/now/`, [Online; Last accessed 11-December-2015] (2014).
[15] L. A. Suchman, Plans and situated actions: the problem of human-machine communication, Xerox Corporation, Palo Alto Research Center, 1985.
[16] B. A. Nardi, Studying context: A comparison of activity theory, situated action models, and distributed cognition, Context and consciousness: Activity theory and human-computer interaction (1996) 69–102.
[17] S. Greenberg, Context as a dynamic construct, Human-Computer Interaction 16 (2) (2001) 257–268.

[18] N. V. Flor, E. L. Hutchins, A case study of team programming during perfective software maintenance, in: Empirical studies of programmers: Fourth workshop, Intellect Books, 1991, p. 36.
[19] G. Fitzpatrick, The locales framework: understanding and designing for wicked problems, Vol. 1, Springer Science & Business Media, 2003.
[20] D. S. Modha, R. Ananthanarayanan, S. K. Esser, A. Ndirango, A. J. Sherbondy, R. Singh, Cognitive computing, Communications of the ACM 54 (8) (2011) 62–71.
[21] H. Dreyfus, What Computers can't do. The Limit of Artificial Intelligence. Revised edition, Harper and Row, New York a.o., 1979.
[22] H. L. Dreyfus, Intelligence without representation–merleau-ponty's critique of mental representation the relevance of phenomenology to scientific explanation, Phenomenology and the Cognitive Sciences 1 (4) (2002) 367–383.
[23] J. R. Lucas, Minds, machines and gödel, Philosophy 36 (137) (1961) 112–127.
[24] J. R. Searle, Minds, brains, and programs, Behavioral and brain sciences 3 (03) (1980) 417–424.
[25] J. Fodor, The Mind Doesn't Work that Way: The Scope and Limits of Computational Psychology, Bradford book, MIT Press, 2001.
[26] A. K. Dey, Providing architectural support for building context-aware applications, Ph.D. thesis, Georgia Institute of Technology (2000).
[27] B. Hardian, J. Indulska, K. Henricksen, Balancing autonomy and user control in context-aware systems-a survey, in: Pervasive Computing and Communications Workshops, 2006. PerCom Workshops 2006. Fourth Annual IEEE International Conference on, IEEE, 2006, pp. 6–pp.
[28] L. Barkhuus, A. Dey, Is context-aware computing taking control away from the user? three levels of interactivity examined, in: UbiComp 2003: Ubiquitous Computing, Springer, 2003, pp. 149–156.
[29] B. H. Cheng, R. De Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, et al., Software engineering for self-adaptive systems: A research roadmap, in: Software engineering for self-adaptive systems, Springer, 2009, pp. 1–26.
[30] M. Salehie, L. Tahvildari, Self-adaptive software: Landscape and research challenges, ACM Transactions on Autonomous and Adaptive Systems (TAAS) 4 (2) (2009) 14.
[31] R. Mizouni, M. A. Matar, Z. Al Mahmoud, S. Alzahmi, A. Salah, A framework for context-aware self-adaptive mobile applications spl, Expert Systems with applications 41 (16) (2014) 7549–7564.
[32] S. Hallsteinsen, K. Geihs, N. Paspallis, F. Eliassen, G. Horn, J. Lorenzo, A. Mamelli, G. A. Papadopoulos, A development framework and methodology for self-adapting applications in ubiquitous computing environments, Journal of Systems and Software 85 (12) (2012) 2840–2859.
[33] R. Rouvoy, P. Barone, Y. Ding, F. Eliassen, S. Hallsteinsen, J. Lorenzo, A. Mamelli, U. Scholz, Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments, in: Software engineering for self-adaptive systems, Springer, 2009, pp. 164–182.
[34] K. Geihs, M. Wagner, Context-awareness for self-adaptive applications in ubiquitous computing environments, in: Context-Aware Systems and Applications, Springer, 2013, pp. 108–120.
[35] G. Chen, D. Kotz, et al., A survey of context-aware mobile computing research, Tech. rep., Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College (2000).
[36] A. K. Dey, A. Newberger, Support for context-aware intelligibility and control, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, 2009, pp. 859–868.
[37] B. Y. Lim, A. K. Dey, Assessing demand for intelligibility in context-aware applications, in: Proceedings of the 11th international conference on Ubiquitous computing, ACM, 2009, pp. 195–204.
[38] B. Y. Lim, A. K. Dey, Toolkit to support intelligibility in context-aware applications, in: Proceedings of the 12th ACM international conference on Ubiquitous computing, ACM, 2010, pp. 13–22.
[39] M. Mori, A software lifecycle process for context-aware adaptive systems, in: Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, ACM, 2011, pp. 412–415.
[40] P. Inverardi, M. Mori, A software lifecycle process to support consistent evolutions, in: Software Engineering for Self-Adaptive Systems II, Springer, 2013, pp. 239–264.
[41] M. Mori, F. Li, C. Dorn, P. Inverardi, S. Dustdar, Leveraging state-based user preferences in context-aware reconfigurations for self-adaptive systems, in: Software Engineering and Formal Methods, Springer, 2011, pp. 286–301.
[42] A. Aztiria, J. C. Augusto, R. Basagoiti, A. Izaguirre, D. J. Cook, Learning frequent behaviors of the users in intelligent environments, Systems, Man, and Cybernetics: Systems, IEEE Transactions on 43 (6) (2013) 1265–1278.
[43] U. Alegre, J. C. Augusto, A. Aztiria, Temporal reasoning for intuitive specification of context-awareness, in: Intelligent Environments (IE), 2014 International Conference on, IEEE, 2014, pp. 234–241.
[44] H. Lieberman, F. Paternò, M. Klann, V. Wulf, End-User Development: An Emerging Paradigm, Vol. 9 of Human-Computer Interaction Series, Springer Netherlands, Dordrecht, 2006.
[45] B. Ur, E. McManus, M. Pak Yong Ho, M. L. Littman, Practical trigger-action programming in the smart home, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, 2014, pp. 803–812.
[46] J. Huang, M. Cakmak, Supporting mental model accuracy in trigger-action programming, in: Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing, ACM, 2015, pp. 215–225.
[47] A. K. Dey, T. Sohn, S. Streng, J. Kodama, icap: Interactive prototyping of context-aware applications, in: Pervasive Computing, Springer, 2006, pp. 254–271.
[48] A. Newberger, A. K. Dey, Designer support for context monitoring and control, IRB-TR-03-017, Intel Research Berkeley (2003).
[49] A. K. Dey, G. D. Abowd, D. Salber, A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications, Human-computer interaction 16 (2) (2001) 97–166.
[50] M. Ball, V. Callaghan, M. Gardner, An adjustable-autonomy agent for intelligent environments, in: Intelligent Environments (IE), 2010 Sixth International Conference on, IEEE, 2010, pp. 1–6.
[51] G. Fischer, Context-aware systems: the'right'information, at the'right'time, in the'right'place, in the'right'way, to the'right'person, in: Proceedings of the International Working Conference on Advanced Visual Interfaces, ACM, 2012, pp. 287–294.
[52] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, et al., Software engineering for self-adaptive systems: A second research roadmap, in: Software Engineering for Self-Adaptive Systems II, Springer, 2013, pp. 1–32.
[53] B. Y. Lim, Improving trust in context-aware applications with intelligibility, in: Proceedings of the 12th ACM international conference adjunct papers on Ubiquitous computing-Adjunct, ACM, 2010, pp. 477–480.
[54] G. Fischer, End-user development and meta-design: Foundations for cultures of participation, in: End-user development, Springer, 2009, pp. 3–14.
[55] J. Pascoe, Adding generic contextual capabilities to wearable computers, in: Wearable Computers, 1998. Digest of Papers. Second International Symposium on, IEEE, 1998, pp. 92–99.
[56] D. Kramer, J. C. Augusto, T. Clark, Context-awareness to increase inclusion of people with ds in society, in: Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence, 2014.
[57] G. Ghiani, M. Manca, F. Paternò, C. Porta, Beyond responsive design: context-dependent multimodal augmentation of web applications, in: Mobile Web Information Systems, Springer, 2014, pp. 71–85.
[58] C. L. T. Yuan, D. A. Ramli, Frog sound identification system for frog species recognition, in: Context-Aware Systems and Applications, Springer, 2013, pp. 41–50.
[59] D. Grassi, A. Bouhtouch, G. Cabri, Inbooki: Context-aware adaptive e-books, in: Context-Aware Systems and Applications, Springer, 2014, pp. 57–66.
[60] A. K. Dey, G. Kortuem, D. R. Morse, A. Schmidt, Situated interaction and context-aware computing, Personal and Ubiquitous Computing 5 (1) (2001) 1–3.
[61] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, D. Riboni, A survey of context modelling and reasoning techniques, Pervasive and Mobile Computing 6 (2) (2010) 161–180.
[62] K. Henricksen, J. Indulska, Modelling and using imperfect context information, in: Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on, IEEE,

2004, pp. 33–37.
[63] J.-y. Hong, E.-h. Suh, S.-J. Kim, Context-aware systems: A literature review and classification, Expert Systems with Applications 36 (4) (2009) 8509–8522.
[64] M. Perttunen, J. Riekki, O. Lassila, Context representation and reasoning in pervasive computing: a review, International Journal of Multimedia and Ubiquitous Engineering 4 (4) (2009) 1–28.
[65] M. Jonsson, Sensing and making sense: Designing middleware for context aware computing, Ph.D. thesis, The Royal Institute of Technology School (2007).
[66] A. Bikakis, T. Patkos, G. Antoniou, D. Plexousakis, A survey of semantics-based approaches for context reasoning in ambient intelligence, in: Constructing ambient intelligence, Springer, 2008, pp. 14–23.
[67] R. Brachman, H. Levesque, Knowledge representation and reasoning, Elsevier, 2004.
[68] S. Jones, S. Hara, J. Augusto, e-friend: an ethical framework for intelligent environment development, in: Ethics and Information Technology, Vol. 17, Springer, 2015, pp. 11–25.
[69] T. Buchholz, A. Küpper, M. Schiffers, Quality of context: What it is and why we need it, in: Proceedings of the workshop of the HP OpenView University Association, Vol. 2003, 2003, pp. pp–32.
[70] J. Pascoe, N. Ryan, D. Morse, Issues in developing context-aware computing, in: Handheld and ubiquitous computing, Springer, 1999, pp. 208–221.
[71] S. Chetan, A. Ranganathan, R. Campbell, Towards fault tolerance pervasive computing, Technology and Society Magazine, IEEE 24 (1) (2005) 38–44.
[72] W. K. Edwards, V. Bellotti, A. K. Dey, M. W. Newman, The challenges of user-centered design and evaluation for infrastructure, in: Proceedings of the SIGCHI conference on Human factors in computing systems, ACM, 2003, pp. 297–304.
[73] K. Pohl, Requirements engineering: fundamentals, principles, and techniques, Springer Publishing Company, Incorporated, 2010.
[74] B. Nuseibeh, S. Easterbrook, Requirements engineering: a roadmap, in: Proceedings of the Conference on the Future of Software Engineering, ACM, 2000, pp. 35–46.
[75] S. Robertson, J. Robertson, Mastering the requirements process: getting requirements right, Addison-Wesley, 2012.
[76] D. Zowghi, C. Coulin, Requirements elicitation: A survey of techniques, approaches, and tools, in: Engineering and managing software requirements, Springer, 2005, pp. 19–46.
[77] J. Krogstie, Requirement engineering for mobile information systems, in: Proceedings of the seventh international workshop on requirements engineering: Foundations for software quality (REFSQ'01), 2001.
[78] D. Hong, D. K. Chiu, V. Y. Shen, Requirements elicitation for the design of context-aware applications in a ubiquitous environment, in: Proceedings of the 7th international conference on Electronic commerce, ACM, 2005, pp. 590–596.
[79] L. Kolos-Mazuryk, G. J. Poulisse, P. A. T. van Eck, Requirements engineering for pervasive services, in: Second Workshop on Building Software for Pervasive Computing. Position Papers., San Diego, California, USA, 2005, pp. 18–22.
[80] A. Sutcliffe, S. Fickas, M. M. Sohlberg, Pc-re: a method for personal and contextual requirements engineering with some experience, Requirements Engineering 11 (3) (2006) 157–173.
[81] K. E. Kjær, Ethnographic studies as a requirement gathering process for the design of context aware middleware, in: Proceedings of the 2007 ACM/IFIP/USENIX international conference on Middleware companion, ACM, 2007, p. 3.
[82] C. Evans, L. Brodie, J. C. Augusto, Requirements engineering for intelligent environments, in: Intelligent Environments (IE), 2014 International Conference on, IEEE, 2014, pp. 154–161.
[83] B. Desmet, J. Vallejos, P. Costanza, W. De Meuter, T. DHondt, Context-oriented domain analysis, in: Modeling and Using Context, Springer, 2007, pp. 178–191.
[84] J. Choi, Context-driven requirements analysis, in: Computational Science and Its Applications–ICCSA 2007, Springer, 2007, pp. 739–748.
[85] J. Choi, Y. Lee, Use-case driven requirements analysis for context-aware systems, in: Computer Applications for Bio-technology, Multimedia, and Ubiquitous City, Springer, 2012, pp. 202–209.
[86] T. Ruiz-López, C. Rodríguez-Domínguez, M. J. Rodríguez, S. F. Ochoa, J. L. Garrido, Context-aware self-adaptations: From requirements specification to code generation, in: Ubiquitous Computing and Ambient Intelligence. Context-Awareness and Context-Driven Interaction, Springer, 2013, pp. 46–53.
[87] W. Sitou, B. Spanfelner, Towards requirements engineering for context adaptive systems, in: Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International, Vol. 2, IEEE, 2007, pp. 593–600.
[88] A. Finkelstein, A. Savigni, A framework for requirements engineering for context-aware services, in: In Proc. of 1 st International Workshop From Software Requirements to Architectures (STRAW), 2001, pp. 200–1.
[89] K. Oyama, H. Jaygarl, J. Xia, C. K. Chang, A. Takeuchi, H. Fujimoto, Requirements analysis using feedback from context awareness systems, in: Computer Software and Applications, 2008. COMPSAC'08. 32nd Annual IEEE International, IEEE, 2008, pp. 625–630.
[90] L. Baresi, L. Pasquale, P. Spoletini, Fuzzy goals for requirements-driven adaptation, in: Requirements Engineering Conference (RE), 2010 18th IEEE International, IEEE, 2010, pp. 125–134.
[91] S. H. Siadat, M. Song, Understanding requirement engineering for context-aware service-based applications, Journal of Software Engineering and Applications 5 (8) (2012) 536–544.
[92] D. Preuveneers, P. Novais, A survey of software engineering best practices for the development of smart applications in ambient intelligence, Journal of Ambient Intelligence and Smart Environments 4 (3) (2012) 149–162.
[93] J. S. Bauer, M. W. Newman, J. A. Kientz, What designers talk about when they talk about context, Human–Computer Interaction 29 (5-6) (2014) 420–450.
[94] T. Winograd, Architectures for context, Human-Computer Interaction 16 (2) (2001) 401–419.
[95] H. Chen, An intelligent broker architecture for pervasive context-aware systems, Ph.D. thesis, University of Maryland (2004).
[96] M. Baldauf, S. Dustdar, F. Rosenberg, A survey on context-aware systems, International Journal of Ad Hoc and Ubiquitous Computing 2 (4) (2007) 263–277.
[97] R. Taylor, N. Medvidovic, E. Dashofy, Software Architecture: Foundations, Theory, and Practice, Wiley, 2009.
[98] P. D. Costa, L. F. Pires, M. van Sinderen, Architectural patterns for context-aware services platforms, in: Ubiquitous Computing, Proceedings of the 2nd International Workshop on Ubiquitous Computing, IWUC 2005, In conjunction with ICEIS 2005, Miami, USA, May 2005, 2005, pp. 3–18.
[99] I. Khalil, Handbook of research on mobile multimedia, IGI Global, 2008.
[100] D. Cassou, Développement logiciel orienté paradigme de conception: la programmation dirigée par la spécification, Ph.D. thesis, Université Sciences et Technologies-Bordeaux I (2011).
[101] L. Daniele, P. D. Costa, L. F. Pires, Towards a rule-based approach for context-aware applications, in: Dependable and Adaptable Networks and Services, Springer, 2007, pp. 33–43.
[102] P. Patel, B. Morin, S. Chaudhary, A model-driven development framework for developing sense-compute-control applications, in: Proceedings of the 1st International Workshop on Modern Software Engineering Methods for Industrial Automation, ACM, 2014, pp. 52–61.
[103] K. E. Kjær, A survey of context-aware middleware, in: Proceedings of the 25th conference on IASTED International Multi-Conference: Software Engineering, ACTA Press, 2007, pp. 148–155.
[104] K. Henricksen, J. Indulska, T. McFadden, S. Balasubramaniam, Middleware for distributed context-aware systems, in: On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE, Springer, 2005, pp. 846–863.
[105] Y. Mor, N. Winters, Design approaches in technology-enhanced learning, Interactive Learning Environments 15 (1) (2007) 61–75.
[106] G. Salvaneschi, C. Ghezzi, M. Pradella, Context-oriented programming: A software engineering perspective, Journal of Systems and Software 85 (8) (2012) 1801–1817.
[107] A. J. Ramirez, B. H. Cheng, Design patterns for developing dynamically adaptive systems, in: Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, ACM, 2010, pp. 49–58.

[108] E. S. Chung, J. I. Hong, J. Lin, M. K. Prabaker, J. A. Landay, A. L. Liu, Development and evaluation of emerging design patterns for ubiquitous computing, in: Proceedings of the 5th conference on Designing interactive systems: processes, practices, methods, and techniques, ACM, 2004, pp. 233–242.

[109] J. A. Landay, G. Borriello, Design patterns for ubiquitous computing, Computer 36 (8) (2003) 93–95.

[110] G. Rossi, S. Gordillo, F. Lyardet, Design patterns for context-aware adaptation, in: Applications and the Internet Workshops, 2005. Saint Workshops 2005. The 2005 Symposium on, IEEE, 2005, pp. 170–173.

[111] O. Riva, C. Di Flora, S. Russo, K. Raatikainen, Unearthing design patterns to support context-awareness, in: Pervasive Computing and Communications Workshops, 2006. PerCom Workshops 2006. Fourth Annual IEEE International Conference on, IEEE, 2006, pp. 5–pp.

[112] E. W. Dijkstra, The humble programmer, Communications of the ACM 15 (10) (1972) 859–866.

[113] J. C. Augusto, Increasing reliability in the development of intelligent environments, in: Intelligent Environments 2009: Proceedings of the 5th International Conference on Intelligent Environments, Barcelona 2009, Vol. 2, IOS Press, 2009, p. 134.

[114] J. C. Augusto, H. Zheng, M. Mulvenna, H. Wang, W. Carswell, P. Jeffers, Design and modelling of the nocturnal aal care system, in: Ambient Intelligence-Software and Applications, Springer, 2011, pp. 109–116.

[115] J. C. Augusto, M. J. Hornos, Software simulation and verification to increase the reliability of intelligent environments, Advances in Engineering Software 58 (2013) 18–34.

[116] G. J. Holzmann, The model checker spin, IEEE Transactions on software engineering 23 (5) (1997) 279–295.

[117] D. Preuveneers, Y. Berbers, Consistency in context-aware behavior: a model checking approach., in: Intelligent Environments (Workshops), 2012, pp. 401–412.

[118] L. DErrico, M. Loreti, Context aware specification and verification of distributed systems, in: Trustworthy Global Computing, Springer, 2012, pp. 142–159.

[119] R. De Nicola, G. L. Ferrari, R. Pugliese, Klaim: A kernel language for agents interaction and mobility, Software Engineering, IEEE Transactions on 24 (5) (1998) 315–330.

[120] Y. Liu, C. Xu, S. Cheung, Afchecker: Effective model checking for context-aware adaptive applications, Journal of Systems and Software 86 (3) (2013) 854–867.

[121] M. Sama, S. Elbaum, F. Raimondi, D. S. Rosenblum, Z. Wang, Context-aware adaptive applications: Fault patterns and their automated identification, Software Engineering, IEEE Transactions on 36 (5) (2010) 644–661.

[122] R. Gerth, Concise promela reference, 1997, URL http://cm. bell-labs. com/cm/cs/what/spin/Man/Quick. html.

[123] J. C. Augusto, M. J. Hornos, Designing more reliable mas-based ambient intelligence systems., in: T. Bosse (Ed.), Agents and Ambient Intelligence, Vol. 12 of Ambient Intelligence and Smart Environments, IOS Press, 2012, pp. 65–90.

[124] R. De Nicola, M. Loreti, Momo: A modal logic for reasoning about mobility, in: Formal Methods for Components and Objects, Springer, 2005, pp. 95–119.

[125] J. Park, M. Moon, S. Hwang, K. Yeom, Cass: A context-aware simulation system for smart home, in: Software Engineering Research, Management & Applications, 2007. SERA 2007. 5th ACIS International Conference on, IEEE, 2007, pp. 461–467.

[126] Z. Wang, S. Elbaum, D. S. Rosenblum, Automated generation of context-aware tests, in: Software Engineering, 2007. ICSE 2007. 29th International Conference on, IEEE, 2007, pp. 406–415.

[127] B. Bertran, J. Bruneau, D. Cassou, N. Loriant, E. Balland, C. Consel, Diasuite: A tool suite to develop sense/compute/control applications, Science of Computer Programming 79 (2014) 39–51.

[128] J. Bruneau, W. Jouve, C. Consel, Diasim: A parameterized simulator for pervasive computing applications, in: Mobile and Ubiquitous Systems: Networking & Services, MobiQuitous, 2009. MobiQuitous' 09. 6th Annual International, IEEE, 2009, pp. 1–10.

[129] L. Yu, W. T. Tsai, Y. Jiang, J. Gao, Generating test cases for context-aware applications using bigraphs, in: Software Security and Reliability, 2014 Eighth International Conference on, IEEE, 2014, pp. 137–146.

[130] T. Strang, C. Linnhoff-Popien, A context modeling survey, in: In: Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England, Springer-Verlag, 2004, pp. 31–41.

[131] C. Bolchini, C. A. Curino, E. Quintarelli, F. A. Schreiber, L. Tanca, A data-oriented survey of context models, SIGMOD Rec. 36 (4) (2007) 19–26.

[132] J. Indulska, P. Sutton, Location management in pervasive systems, in: Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003-Volume 21, Australian Computer Society, Inc., 2003, pp. 143–151.

[133] A. Fortier, N. Canibano, J. Grigera, G. Rossi, S. Gordillo, An object-oriented approach for context-aware applications, in: Advances in Smalltalk, Springer, 2007, pp. 23–46.

[134] D. Graff, M. Werner, H. Parzyjegla, J. Richling, G. Mühl, An object-oriented and context-aware approach for distributed mobile applications, in: Architecture of Computing Systems (ARCS), 2010 23rd International Conference on, VDE, 2010, pp. 1–10.

[135] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, J. Irwin, Aspect-oriented programming, Springer, 1997.

[136] É. Tanter, K. Gybels, M. Denker, A. Bergel, Context-aware aspects, in: Software Composition, Springer, 2006, pp. 227–242.

[137] F. Dantas, T. Batista, N. Cacho, Towards aspect-oriented programming for context-aware systems: A comparative study, in: Proceedings of the 1st International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments, IEEE Computer Society, 2007, p. 4.

[138] L. Fuentes, N. Gámez, P. Sánchez, Aspect-oriented design and implementation of context-aware pervasive applications, ISSE 5 (1) (2009) 79–93.

[139] L. Fuentes, N. Gamez, P. Sanchez, Aspect-oriented executable uml models for context-aware pervasive applications, in: Model-based Methodologies for Pervasive and Embedded Software, 2008. MOMPES 2008. 5th International Workshop on, IEEE, 2008, pp. 34–43.

[140] S. Apel, The role of features and aspects in software development, Ph.D. thesis, Otto-von-Guericke-Universität Magdeburg, Universitätsbibliothek (2007).

[141] R. Evans, P. Kearney, G. Caire, F. Garijo, J. Gomez Sanz, J. Pavon, F. Leal, P. Chainho, P. Massonet, Message: Methodology for engineering systems of software agents, EURESCOM, EDIN (2001) 0223–0907.

[142] R. Hirschfeld, P. Costanza, O. Nierstrasz, Context-oriented programming, Journal of Object Technology, ETH Zurich, 7 (3) (2008) 125–151.

[143] V. R. Lesser, Cooperative multiagent systems: A personal view of the state of the art, Knowledge and Data Engineering, IEEE Transactions on 11 (1) (1999) 133–142.

[144] P. Heymans, J.-C. Trigaux, Software product line: state of the art, Relatório Técnico EPH3310300R0462/215315, Product Line ENgineering of food TraceabilitY software (PLENTY) Project, Institut dInformatique, FUNDP, Namur.

[145] A. Sturm, O. Shehory, Agent-oriented software engineering: Revisiting the state of the art, in: Agent-Oriented Software Engineering, Springer, 2014, pp. 13–26.

[146] M. Appeltauer, R. Hirschfeld, M. Haupt, J. Lincke, M. Perscheid, A comparison of context-oriented programming languages, in: International Workshop on Context-Oriented Programming, ACM, 2009, p. 6.

[147] S. Apel, C. Kästner, An overview of feature-oriented software development., Journal of Object Technology 8 (5) (2009) 49–84.

[148] N. Ubayashi, S. Nakajima, Context-aware feature-oriented modeling with an aspect extension of VDM, in: Proceedings of the 2007 ACM Symposium on Applied Computing (SAC), Seoul, Korea, March 11-15, 2007, 2007, pp. 1269–1274.

[149] D. Kramer, Unified gui adaptation in dynamic software product lines, Ph.D. thesis, University of West London (2014).

[150] P. Fernandes, C. Werner, L. G. P. Murta, Feature modeling for context-aware software product lines., in: The 20th International Conference on Software Engineering and Knowledge Engineering (SEKE), 2008, pp. 758–763.

[151] P. Fernandes, C. Werner, E. Teixeira, An approach for feature modeling of context-aware software product line., Journal of Universal Computer Science (JUCS) 17 (5) (2011) 807–829.

[152] C. Parra, X. Blanc, L. Duchien, Context awareness for dynamic service-oriented product lines, in: Proceedings of the 13th International Soft-

ware Product Line Conference, Carnegie Mellon University, 2009, pp. 131–140.
[153] F. G. Marinho, R. Andrade, C. Werner, W. Viana, M. E. Maia, L. S. Rocha, E. Teixeira, V. L. Dantas, F. Lima, S. Aguiar, et al., Mobiline: A nested software product line for the domain of mobile and context-aware applications, Science of Computer Programming 78 (12) (2013) 2381–2398.
[154] F. G. Marinho, R. M. Andrade, C. Werner, A verification mechanism of feature models for mobile and context-aware software product lines, in: Software Components, Architectures and Reuse (SBCARS), 2011 Fifth Brazilian Symposium on, IEEE, 2011, pp. 1–10.
[155] D. Kramer, S. Oussena, P. Komisarczuk, T. Clark, Using document-oriented guis in dynamic software product lines, in: ACM SIGPLAN Notices, ACM, 2013, pp. 85–94.
[156] D. Kramer, A. Kocurova, S. Oussena, T. Clark, P. Komisarczuk, An extensible, self contained, layered approach to context acquisition, in: Proceedings of the Third International Workshop on Middleware for Pervasive Mobile and Embedded Computing, ACM, 2011, p. 6.
[157] G. M. Kapitsaki, G. N. Prezerakos, N. D. Tselikas, I. S. Venieris, Context-aware service engineering: A survey, Journal of Systems and Software 82 (8) (2009) 1285–1297.
[158] D. B. Abeywickrama, Context-aware services engineering for service-oriented architectures, in: Web Services Foundations, Springer, 2014, pp. 291–317.
[159] O. M. Shehory, A. Sturm, A brief introduction to agents, in: Agent-oriented software engineering: reflections on architectures, methodologies, languages, and frameworks, Springer, 2014, pp. 3–3.
[160] T. Bosse, Agents and Ambient Intelligence: Achievements and Challenges in the Intersection of Agent Technology and Ambient Intelligence, IOS Press, Amsterdam, The Netherlands, The Netherlands, 2012.
[161] Y. Shoham, An overview of agent-oriented programming, Software agents 4.
[162] M. Wooldridge, An introduction to multiagent systems, John Wiley & Sons, 2009.
[163] A. R. d. M. Neves, Á. M. G. Carvalho, C. G. Ralha, Agent-based architecture for context-aware and personalized event recommendation, Expert Systems with Applications 41 (2) (2014) 563–573.
[164] P. K. Murukannaiah, M. P. Singh, Xipho: Extending tropos to engineer context-aware personal agents, in: Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems, International Foundation for Autonomous Agents and Multiagent Systems, 2014, pp. 309–316.
[165] J. L. Barbosa, A. C. Yamin, P. Vargas, I. Augustin, C. F. Geyer, Holoparadigm: a multiparadigm model oriented to development of distributed systems, in: Parallel and Distributed Systems, 2002. Proceedings. Ninth International Conference on, IEEE, 2002, pp. 165–170.
[166] A. C. Yamin, J. V. Barbosa, I. Augustin, L. C. da Silva, R. Real, C. Geyer, G. Cavalheiro, Towards merging context-aware, mobile and grid computing, International Journal of High Performance Computing Applications 17 (2) (2003) 191–203.
[167] J. Barbosa, F. Dillenburg, G. Lermen, A. Garzao, C. Costa, J. Rosa, Towards a programming model for context-aware applications, Computer Languages, Systems & Structures 38 (3) (2012) 199–213.
[168] J. Miller, J. Mukerji, Mda guide version 1.0.1, Tech. rep., Object Management Group (OMG) (2003).
[169] B. Selic, The pragmatics of model-driven development, IEEE software 20 (5) (2003) 19–25.
[170] Q. Z. Sheng, B. Benatallah, Contextuml: a uml-based modeling language for model-driven development of context-aware web services, in: Mobile Business, 2005. ICMB 2005. International Conference on, IEEE, 2005, pp. 206–212.
[171] J. Rumbaugh, I. Jacobson, G. Booch, The Unified Modeling Language Reference Manual, Addison-Wesley Professional, 2005.
[172] E. Serral, P. Valderas, V. Pelechano, A model driven development method for developing context-aware pervasive systems, in: Ubiquitous Intelligence and Computing, Springer, 2008, pp. 662–676.
[173] E. Serral, P. Valderas, V. Pelechano, Towards the model driven development of context-aware pervasive systems, Pervasive and Mobile Computing 6 (2) (2010) 254–280.
[174] R. Tesoriero, J. A. Gallud, M. D. Lozano, V. M. R. Penichet, Cauce: Model-driven development of context-aware applications for ubiquitous computing environments., Journal of Universal Computer Science 16 (15) (2010) 2111–2138.
[175] A. Sindico, V. Grassi, Model driven development of context aware software systems, in: International workshop on context-oriented programming, ACM, 2009, p. 7.
[176] J. R. Hoyos, J. García-Molina, J. A. Botía, Mlcontext: a context-modeling language for context-aware systems, Electronic Communications of the EASST 28 (2010).
[177] A. A. Nacci, B. Balaji, P. Spoletini, R. Gupta, D. Sciuto, Y. Agarwal, Buildingrules: a trigger-action based system to manage complex commercial buildings, in: Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers, ACM, 2015, pp. 381–384.
[178] G. Lucci, F. Paternò, Understanding end-user development of context-dependent applications in smartphones, in: Human-Centered Software Engineering, Springer, 2014, pp. 182–198.
[179] B. P. Lientz, E. B. Swanson, Software maintenance management, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1980.
[180] D. Moos, S. Bader, T. Kirste, From intelligibility to debuggability in context-aware systems, in: KI 2014: Advances in Artificial Intelligence, Springer, 2014, pp. 219–224.
[181] D. Kulkarni, A. Tripathi, A framework for programming robust context-aware applications, Software Engineering, IEEE Transactions on 36 (2) (2010) 184–197.
[182] I. Augustin, A. C. Yamin, L. C. da Silva, R. A. Real, G. Frainer, C. F. Geyer, Isamadapt: abstractions and tools for designing general-purpose pervasive applications, Software: Practice and Experience 36 (11-12) (2006) 1231–1256.
[183] K. Henricksen, J. Indulska, Developing context-aware pervasive computing applications: Models and approach, Pervasive and mobile computing 2 (1) (2006) 37–64.
[184] B. Guo, D. Zhang, M. Imai, Toward a cooperative programming framework for context-aware applications, Personal and Ubiquitous Computing 15 (3) (2011) 221–233.
[185] A. Moon, H. Kim, H. Kim, S. Lee, Context-aware active services in ubiquitous computing environments, ETRI journal 29 (2) (2007) 169–178.
[186] A. Achilleos, K. Yang, N. Georgalas, Context modelling and a context-aware framework for pervasive service creation: A model-driven approach, Pervasive and Mobile Computing 6 (2) (2010) 281–296.
[187] T. Gross, Towards a new human-centred computing methodology for cooperative ambient intelligence, Journal of Ambient Intelligence and Humanized Computing 1 (1) (2010) 31–42.
[188] L. Tang, Z. Yu, H. Wang, X. Zhou, Z. Duan, Methodology and tools for pervasive application development, International Journal of Distributed Sensor Networks 2014.
[189] J. Baek Jorgensen, C. Bossen, Executable use cases: requirements for a pervasive health care system, Software, IEEE 21 (2) (2004) 34–41.
[190] F. Pérez, P. Valderas, Allowing end-users to actively participate within the elicitation of pervasive system requirements through immediate visualization, in: Requirements Engineering Visualization (REV), 2009 Fourth International Workshop on, IEEE, 2009, pp. 31–40.
[191] R. Fuentes-Fernández, J. J. Gómez-Sanz, J. Pavón, Understanding the human context in requirements elicitation, Requirements engineering 15 (3) (2010) 267–283.
[192] P. A. Laplante, What every engineer should know about software engineering, CRC Press, 2007.